N64 11331

UDE-1

NASA CR 52866

125p.

Final Report

DEVELOPMENT OF TECHNIQUES FOR IMPROVING THE RELIABILITY OF DIGITAL SYSTEMS THROUGH LOGICAL REDUNDANCY

Prepared for:

JET PROPULSION LABORATORIES
4800 OAK GROVE DRIVE
PASADENA, CALIFORNIA

CONTRACT M-44501 UNDER NASA
CONTRACT NASw-6

By: Jack Goldberg, Robert C. Minnick, William H. Kautz

STANFORD RESEARCH INSTITUTE

MENLO PARK, CALIFORNIA

SRI

STANFORD RESEARCH INSTITUTE

MENLO PARK, CALIFORNIA

SRI

May 1962

Final Report

t' DEVELOPMENT OF TECHNIQUES FOR IMPROVING THE RELIABILITY OF DIGITAL SYSTEMS THROUGH LOGICAL REDUNDANCY Final ...,

Prepared for:

JET PROPULSION LABORATORIES
4800 OAK GROVE DRIVE
PASADENA, CALIFORNIA

CONTRACT M-44501 UNDER NASA
(NASA CONTRACT NASw-6 and JPL-M...)

By: *Jack Goldberg, Robert C. Minnick,* and *William H. Kautz* May 1962 125p refs

SRI Project No. 3196

(NASA CR-52866; OTS:

Approved:

J. R. Anderson
J. R. ANDERSON, MANAGER COMPUTER TECHNIQUES LABORATORY

[illegible]
JERRE D. NOE, DIRECTOR ENGINEERING SCIENCES DIVISION

Copy No. 68

ABSTRACT

11331

This report summarizes the second year of study of techniques for improving digital system reliability through application of logical redundancy. Three major topics are discussed: organization of a computer for efficient use of redundancy techniques, use of redundancy to mask faults in computer memories, and logical redundancy techniques for sequential networks. Each of these three topics is treated in a self-contained part of the report.

AUTHOR

CONTENTS

CONTENTS

ILLUSTRATIONS

TABLES

FOREWORD

This is the final report on the second year of a study of techniques for improving the reliability of digital systems through the application of logical redundancy.

The long-range objectives of this program were stated in Sec. II of the initial proposal (EU 59-139, Nov. 13, 1959), as follows:

(1) To determine the interrelations between, and limits of applicability of, known and suggested techniques for introducing redundancy

(2) To evolve new techniques for introducing redundancy, in order to achieve a greater improvement in reliability for fixed, small redundancy ratios

(3) To derive procedures for the synthesis of economical, redundant digital networks

(4) To evolve a design philosophy for redundant digital networks, to serve as a guide to logic and system designers

The first year's research was concerned mainly with combinational digital networks. The major topics investigated during the second year, just ended, were:

(1) Organization of a computer for the efficient use of redundancy techniques

(2) The use of redundancy to mask faults in computer memories (primarily in access circuits)

(3) Logical Redundancy techniques for sequential networks, specifically fault-detection, fault-masking and fault-detection-and-correction

Parts One and Two of this report present the result of the first two studies, *i.e.*, in computer organization and memory reliability. Part Three of this report summarizes the results of the third study, which was the subject of Technical Reports 1 and 2, issued during the last year.

The three parts may be read independently—each contains a summary of results and recommendations for further study. Generally, the project studies have concentrated on techniques that minimize the amount of redundant equipment needed to correct given numbers of faults. Rather than develop fault correction methods for general digital devices, specific components of digital computing systems were examined. In several cases it was possible to take advantage of inherent redundancies, and to permit replacement of a logical block with one giving equivalent performance but which is more easily protected. Attempts were made to distinguish the types of redundancy most suitable for different kinds of digital circuits and their costs and to determine the size of logical block over which different types of redundancy are best applied.

A similar point of view was taken in the study of systems. It was seen that the complex logical structure of a stored program computer could be simplified, while preserving a reasonable efficiency, so that the number of distinct blocks and interconnecting data paths needing checking could be kept small. Also, it was suggested that "wired-in" control equipment be replaced, as much as possible, by stored program steps, since memory storage is more reliable and more easily checked. An attempt was made to distinguish parts of a computer in which errors were most crucial; specifically, means of protecting the processing of the instruction list itself were investigated.

An important method, especially in the memory and logic areas, was the application of the existing large body of knowledge of error-correcting codes, both in combinational and in sequential circuits. Consideration was given to selection of codes that minimize the total amount of equipment, including both the encoding and decoding equipment. Some attention was given to fault-masking the correction equipment itself. A number of useful codes were found, but the subject remains open.

An important consideration, arising in several contexts, was the need for compatibility of redundancy techniques applied to elements of a system, *e.g.*, within a memory system, in a computer as a whole, and within larger system boundaries. It is suggested that the rather large body of redundancy techniques developed so far should be drawn upon in a design of a hypothetical or actual system, in order to uncover problems of consistency and completeness.

PART ONE
A STUDY OF COMPUTER STRUCTURES WHICH FACILITATE THE APPLICATION OF REDUNDANCY*

I INTRODUCTION

Various techniques have been developed for applying redundancy to improve the reliability of different kinds of computer components—transmission circuits, arithmetic units, etc. These techniques may be applied after a given machine design is completed, but one would expect that a more successful system would result if it were designed with the redundancy techniques in mind. Similarly, knowledge of system problems would aid the developer of redundancy techniques in his choice of problems and in the ranges of performance required.

Accordingly, the two major objectives of the present study were:

(1) Development of a model for a stored-program space-mission computer, to serve as a guide for coordinating various error-correction techniques, and

(2) Development of computer organization design principles which lead to a minimum reliability load—that is, the least amount of equipment which can survive a specified number and kind of failures—and which facilitate application of redundancy.

In the study, plans for four simple computers were drawn according to several criteria to be described. There is a wide range of computation speeds represented, and, of course, the slower, simpler machines give greater promise of reliability (the study included estimates of computation speeds, based on multiplication time).

* by Jack Goldberg.

The point of view taken in the system designs was to allow the smallest amount of equipment which meets expected requirements of speed and flexibility. Assuming the classic separation of functions into specialized blocks for memory, control, and arithmetic, it was considered desirable to control the amount and complexity of equipment in these blocks with consideration to their sizes after suitable redundancy has been applied to each.

This is not the only possible useful approach. For example, the elements of a computer could be distinguished in the classes of switches, memory elements and gain elements, and a study could then be made of the best proportioning of these elements, based on assumptions of their relative expected reliabilities.

Such distinctions are artificial, inasmuch as the reliabilities of operation of the elements or blocks are not mutually independent of their relative and absolute sizes or their system functions. Nevertheless, the distinction does serve to simplify the creative task of machine design, since there is, as yet, no general synthesis procedure for computers.

Analogously with the separation of a machine into functional blocks, the computation itself was considered as having two components, *i.e.*, calculations upon functions, and calculations on the program. Examples of the latter are instruction counting and address calculation. It is argued that the integrity of a program calculation is much more important than the integrity of functional calculation, since a properly operating program can overcome errors in functional calculations; hence, a study was made of machine organizations which offer special protection to the program.

Section II of this part of the report presents the result of a brief study of the use of list processing techniques for program protection. Also included in Section II are some remarks on the application of list methods to storage of time schedules for control events.

Section III, which constitutes the main body of Part One, is a study of some very simple, conventionally organized, general-purpose computers. It attempts to assess the consequences of following certain design biases, which aim to facilitate the application of redundancy to computer systems.

Other interesting system approaches remain to be studied. For example,

(1) Different parts of a program may be calculated in independent computers, with a graded degree of redundancy dependent upon importance

(2) Adaptive repair techniques might be used, *i.e.*, components may be envisioned which, periodically, may be adjusted to some standard of performance by a "maintenance" block,

(3) A system might be realized by assembly of a large number of identical blocks—each containing memory, arithmetic and control behavior with facility for selective replacement of blocks—and so on.

Each of these approaches has unique system design problems.

II PROGRAM PROTECTION AND LIST PROCESSING

A. INTRODUCTION

Recent studies of programming languages have developed the notion of lists and list processing. The following sections will consider some very simple applications of the linked-list concept in computers for space vehicles. Three of the lists found in such computers are of special interest:

(1) The set of instruction lists

(2) A general vector or matrix of operands

(3) The list of times at which certain tasks must be performed.

The first two are important in a scheme for eliminating conventional address calculations.

B. ELIMINATION OF ADDRESS CALCULATIONS

Errors in address calculations may be considered to be even more serious than errors in calculations on operands. The latter can be checked and corrected in various ways, *e.g.*, by asymptotic iterations which tolerate temporary deviations, by repeated calculation of the same number (perhaps in several ways, etc.), but an error in an address calculation may result in irreparable deviation from a program or in the use of false operands in a numerical calculation. The use of list processing techniques in program processing is attractive, since they involve copying words rather than calculating them, and copying is more easily checked and corrected than is calculating.

Precedent already exists for list techniques in instruction stepping, *i.e.*, in the "1 + 1" address organization, in which each instruction word carries the address of the succeeding instruction. Since transfer of control also involves only copying, the need for an instruction counter is completely eliminated. The main difficulty comes from the increased memory storage capacity for the recording of succeeding addresses (known commonly as "links"). Some economies are possible by subdividing the link word into, say, two parts, one part each for coarse and fine addressing. Each part can contain its own error-correcting code. The more significant part could be set up occasionally in a special register, by a special instruction, while only the less significant part, *e.g.*, 5 or 6 bits, would actually be stored in memory.

Stepping through the elements of a matrix is somewhat more complicated, since a given element is on two lists, *i.e.*, a row list and a column list, and at different times the program may wish to step through either. A direct solution is simply to record an item together with two independent link terms. The three units of data could be recorded in three "successive" words. The word "successive," of course, may no longer be taken for granted. One means of achieving the succession is to record the three units of data as a microlist, *i.e.*, each unit contains a link to the next unit, as well as its data word, which is a link of the item to other items on a row or column list. The format for the items $X_{i,j}$ and $X_{i+1,j}$ could be something as in Table 1.1:

TABLE 1.1

FORMAT OF MATRIX DATA LIST

LOCATION	CONTENTS OF MEMORY CELL		
	Flag	Number or Macrolink	Microlink to next Location
L(i,j)	11	$X_{i,j}$	L(i,j)+1
L(i,j)+1	10	L(i+1,j)	L(i,j)+2
L(i,j)+2	10	L(i,j+1)	-----
⋮	⋮	⋮	⋮
L(i+1,j)	11	$X_{i+1,j}$	L(i+1,j)+1
L(i+1,j)+1	10	L(i+2,j)	L(i+1,j)+2
L(i+1,j)+2	10	L(L+1,j+1)	-----

The LOCATION column gives the symbolic address of a memory cell, whose contents are specified in the remaining columns. The "flag" bits distinguish between non-list numbers (00), and the first (11) and successive (10) numbers of a microlist. The "Microlink" serves in stepping through the three cells of a microlist, while the "Macrolink" serves in stepping to other matrix elements.

This scheme triples, at least, the storage requirement for matrices (although the new storage could be in permanent form). One may conceive schemes of economizing on the number of link bits actually included in the memory, by use of special registers for coarse addressing, both for macrolinks and microlinks. This would seem feasible for microlinking, but rather questionable for macrolinking.

One interesting implementation of this scheme would be to use a fixed-store memory with indirect addressing. That is, Table 1.1 could be stored completely in a fixed-store memory, except that instead of the number, X_{ij}, there would be stored the *address* of the number in a variable-store memory, whose size would be no greater than that of a conventional computer's memory. The same scheme may be used for addressing instructions.

The net result of this development is to replace the usual address calculation, *i.e.*, arithmetic operations on numbers in radix notation, by a very primitive kind of counting, in which each number-successor pair appears explicitly in a table. The redeeming feature of this otherwise primitive scheme is that economical error-correcting techniques of very high power may be employed.

A block diagram of a possible implementation of this approach is given in Fig. 1.1 and a table of micro-operations for this organization is given in Table 1.2. About half of the steps are due to the indirect addressing of instructions and operands, in order to allow storage of links in permanent memory. Another source of microprogram length is due to the use of only a single address register, and the operation of only one external register at a time. The block labeled ECE represents error correction equipment. Since all address and operation information must flow through the block, errors in instruction data caused in either memory may be corrected, up to a limit. Equipment for correction of arithmetic errors is not shown. Program control and calculation are accomplished in separate blocks, indicating the possibility of allowing the controller to switch between blocks of standby calculation equipment.

The present treatment has been sketchy, since it was intended primarily to determine the basic feasibility of a machine in which the program control sequence requires only inter-register transfers. A proper evaluation of the idea would require some investigation of the number of program steps and program storage space required for representative computations. The importance of the concept also depends upon the relative expected reliabilities of register transfers and arithmetic operations in a particular circuit technology, and upon the relative costs of adding sufficient redundancy to achieve a given reliability.

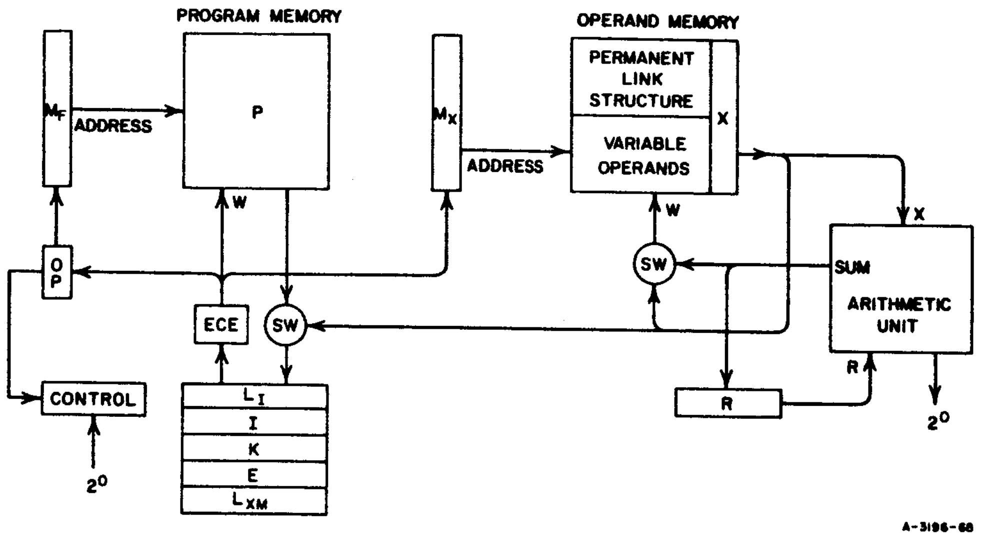


FIG. 1.1 LIST-BASED ORGANIZATION

C. STORAGE OF TIME SCHEDULES

If a time schedule is stored in the form of a table of increments between events, list techniques allow easy insertion or deletion of events without requiring the changing of all time entries. For example, the schedule might be stored in the following form:

Task	Time increment	Link to memory location of next task

TABLE 1.2

MICRO-OPERATIONS FOR ONE PROGRAM STEP IN A LIST-BASED COMPUTER ORGANIZATION

MAJOR PHASES	INTER-REGISTER TRANSFERS	DESCRIPTION OF OPERATION
<u>Fetch:</u>	$M_P \leftarrow L_I$	Set up address of instruction
	$I \leftarrow X$	Fetch instruction
	$L_I \leftarrow X$	Fetch link to next instruction
Option (controlled by instruction bit)		
a) Direct Addressing:	$M_x, OP \leftarrow I$	Set up operation and operand address End Fetch-go to Execute
b) Indirect Addressing:	$M_x, OP \leftarrow I$	or, Set up operation and location of operand address
	$I \leftarrow X$	Fetch operand address
	$M_x \leftarrow I$	Set up operand address (note: if operand is a matrix element, address is that of the running index to the matrix). End Fetch-go to Execute
<u>Execute</u>		
Option (instruction-controlled)		
a) Single operand:	$\left\{\begin{array}{lr} R \leftarrow R/O \pm X & \text{Arith} \\ \text{or } X \leftarrow R + X/O & \text{Store} \\ \text{or } L_I \leftarrow X & \text{Jump} \end{array}\right\}$	End program step
b) Matrix element:	$K \leftarrow X$	Fetch index of current element from index store
	$M_X \leftarrow K$	Set up access to structure table
	$E \leftarrow X$	Fetch location of current element
	$L_{XM} \leftarrow X$	Fetch microlink to next-element link
	$M_x \leftarrow E$	Set up access to element
	$\left\{\begin{array}{lr} R \leftarrow R/O \pm X & \text{Arith} \\ \text{or } X \leftarrow R + X/O & \text{Store} \end{array}\right\}$	<u>End program step</u>
b.1) Link to next row element:		
	$M_x \leftarrow L_{XM}$	Set up access to next row element link
	$L_X \leftarrow X$	Fetch next row-element link
	$M_x \leftarrow I$	Set up access to index store
	$X \leftarrow L_X$	Store next row-element link in matrix index store

TABLE 1.2 *concluded*

b.2) Link to next column element:	$M_x \leftarrow L_{XM}$	**Set up access to next microlink**
	$L_{XM} \leftarrow X$	**Fetch microlink to next column-element link**
	$M_x \leftarrow L_{XM}$	**Set up access to next column-element link**
	$L_X \leftarrow X$	**Fetch next column-element link**
	$M_x \leftarrow I$	**Set up access to index store**
	$X \leftarrow L_X$	**Store next row element link in matrix store.**
		End program step

If a new task is to be inserted, its entry may be recorded anywhere in the memory. Its link joins it to the appropriate successor, and only the preceding entry need be changed. Deletion of a task, and shifting of a portion of the time schedule are similarly simple. The main advantage of this scheme appears to be a reduction in the amount of data needed to be transmitted to effect a change. Its implementation is quite natural in a list-processing computer, but it could also be programmed for a conventionally organized machine.

III SOME SIMPLE COMPUTERS

A. INTRODUCTION

This section examines in some detail the consequences of applying certain approaches in the design of a general purpose computer. The approaches, described below, are intended to facilitate the application of redundancy to a computer system. Four designs are presented, which cover a wide range of speed and complexity, within the general limits of the approaches. Programs for multiplication and division are given in detail in order to assess speed and program storage space, and one logical design is given in some detail, in order to assess equipment costs. The following paragraphs explain the design approaches that guided the sample designs.

Three parameters are useful guides in proportioning the size of memory, arithmetic and control blocks:

(1) present state of the art in reliable construction,
(2) ease of detecting failures, and
(3) ease of correcting failures.

In all three respects, computer memories are superior to the other blocks. The logical selection and storage functions are accomplished by magnetic cores, which have excellent reliability, and the gain-producing elements occur in uniform, grouped structures, which are quite amenable to economical and easily designed error-masking redundancy techniques. The transformation on the data signals is trivial, *i.e.*, delay, so that error-correcting transmission codes may be employed. The transformation on the access selection signals is not trivial, but it is simple.

Arithmetic blocks should probably be rated second, over control blocks, since the rules of arithmetic are simple, and a number of arithmetic error-detecting procedures are known. Arithmetic blocks can probably be reduced in size more easily, since the number of modes, *i.e.*, addition, subtraction, etc., is fewer than the number of control modes usually needed.

An important reason for minimizing arithmetic and control equipment is that provision for fault-masking requires more-or-less triplication of the number of components.[1*] This is not the case with memory, since

*References are grouped at the end of Part One.

(1) the memory elements may not need any duplication, and even if they do, error correcting codes may be used with very modest redundancy ratios,

(2) the drivers may be protected with less than 2:1 redundancy,* and,

(3) while triplication of sense amplifiers may be needed, their number, in the systems described, is small.

The general consequence of these observations is a design emphasis which might be characterized as "Less Logic, More Memory." It should be expected to lead to a reduction in the power and complexity of the individual orders, and an increase in the use of stored subroutines, compared to current general-purpose computers.

Ease of programming definitely should not be a design criterion. The interests of a human or automatic programmer should be completely subordinated to interests of reliability. Two criteria which must be considered, however, are speed of computation, which will be reduced, and physical size of the memory, which will be increased, by the above emphasis.

Another design emphasis is the desire to minimize the number of independent data paths, since it is likely that some error detecting and correcting equipment will be needed to monitor all data flow. Exercising this emphasis will lead to reduced speed.

It appears that presently available memory and logic components are capable of much higher speeds than are needed for space-mission computers. Thus, if the design of a computer requires execution of many steps to accomplish a given task, the consequent slowness of computation could be compensated by using a high clock rate. This will, of course, tend to increase the power consumption (parenthetically, it is not obvious that the net power consumption increases, since the amount of active equipment will be diminished). If, however, reliability is improved significantly by virtue of the reduction and simplification of equipment, compared to a more "efficient" design the increased power may be considered as the price to be paid. Similar remarks apply to increases in memory volume and weight.

In summary, we seek to reduce the amount of logical equipment and the number of data paths at the price of increased memory size and increased programming complexity, within bounds of required computation rate and physical limitations of power, weight, etc.

*See Part Two, this report.

At this point it may be appropriate to refer to some studies which similarly pursued the more-memory-less-logic approach, for theoretical interests in minimality[2], for simplicity in control[3], or for reliability[4]

B. GENERAL MACHINE CHARACTERISTICS AND CONVENTIONAL STRUCTURES.

The machines considered here have characteristics which are expected to be required of a flexible guidance and control computer used in long-term space missions. They are assumed to operate by execution of a stored program capable of general purpose computation. A set of input and output channels are served, some with the provision for independent interruption of normal computation.

For convenience of discussion, numbers are assumed to be in binary representation, normalized to be within the range of -1 to $+(1-2^{-n})$ inclusive, with a bit length, n, of about 20. "Two's complement" representation is assumed, in preference to "signed-absolute," because

(1) in simple addition and subtraction, which are machine orders, there is never a need to recalculate a result for certain magnitude relations— thus saving time (and logical equipment) for the basic operations, and

(2) multiplications, which are slightly more complicated than in "signed-absolute" arithmetic will be executed by subroutine, so that the extra complexity does not result in additional equipment. This subroutine will be discussed in Subsection III-I.

The main working store is assumed to be a static, randomly addressable memory, with fixed word length. A single-address instruction is assumed in order to simplify the Control.

C. A HYPOTHETICAL CONVENTIONAL COMPUTER

In describing the special features of the suggested computer organizations, it may be convenient to compare them with the features of a hypothetical computer which epitomizes conventional organizations. Such a computer is illustrated in Fig. 1.2.

The main components are a random-access memory, driven by an address register, a set of independently selected registers, performing the functions of arithmetic (A_1, A_2, A_3), instruction addressing (IC), instruction storing (OP, Addr), and instruction indexing (B), an arithmetic unit, a control and timing unit, input and output registers and, perhaps, an input priority controller. The interconnections between these components are shown lumped in one block. They comprise serial and parallel data transmissions, some fixed, and some subject to control. All of the arithmetic equipment may be concentrated, as shown, or it may appear at several places, *e.g.*, the instruction address register may also count.

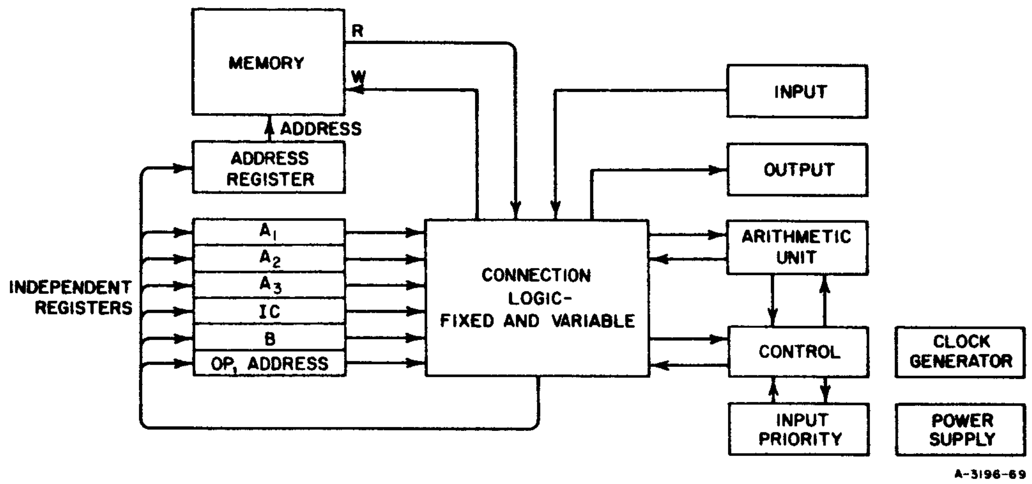


FIG. 1.2 HYPOTHETICAL COMPUTER

Functionally, a conventional computer would be expected to have the operations +, –, x and ÷, in algebraic and absolute variations, perhaps some logical and formating operations, and some special input-output orders, all as built-in machine orders.

D. AUGMENTING THE MEMORY

The computer designs to be described give an important role to the memory unit. Certain properties will be assumed which are unconventional, but which are capable of realization by conventional construction techniques. This subsection will specify the properties of the kind of memory unit assumed in the remainder of the report. Figure 1.3 shows the major functional sections.

The data are assumed to reside in word cells of fixed length. Some data may be permanently recorded. A given word is selected for reading or writing by presenting an address number to a word selection switch, via a set of power drivers. Both for reading and for writing, the individual

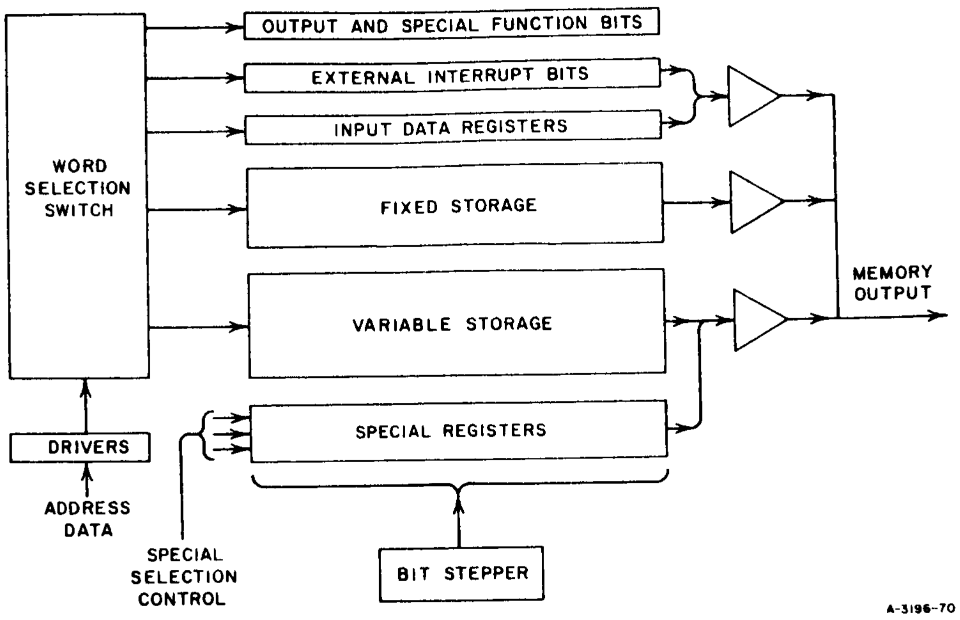


FIG. 1.3 MAJOR FUNCTIONAL SECTIONS OF MEMORY

bits of a selected word are activated serially; thus, in principle, one read amplifier and one write amplifier are needed. The task of successively activating bits is nominally assigned to the block labeled "Bit Stepper."

The selection of individual bits may be accomplished by coincident-current selection, with one current produced on a word-select basis and the other on a bit-select basis. However, the use of coincident-current selection may be undesirable because of the stringent control of current driver and core characteristics required. One good solution appears to be available in the use of multi-aperture memory elements. Several schemes have been described in the literature[5,6] which permit unlimited driving. Furthermore, several schemes may be conceived which do not require that the word- and bit-selection currents for reading occur at the same moment, so that an entire word may be primed at one moment, with its bits being scanned subsequently. Such a scheme has the incidental benefit of freeing the address register during the reading of a memory word, which would avoid the need for separate external registers for instruction address and operand address. The feature of unlimited drive is particularly useful, in that it permits exploitation of load-sharing behavior in selection switches, for compensation of driver failures. This topic is discussed in detail in Part Two of this report. Another good solution to the problem of variation in selected-line output appears to be available in special non-linear current regulators, design of which also is considered in Part Two.

In addition to the words addressable through the selection switch, several other word cells labeled Special Registers are assumed, which may be selected either by the main selection switch, or by special control, *i.e.*, by the computer control circuits. Otherwise, the bit selection and the reading and writing circuits are presumed to be common to the memory proper. These registers will be used for arithmetic and address manipulation, to replace the special external registers illustrated in the hypothetical computer of Fig. 1-2. The over-all computer reliability would hopefully be greater, since the registers would not add appreciably to the load of sense amplifiers and bit sequencing drivers, which must be provided in either case.

In Scheme I designs below, this advantage will be purchased by a decrease in speed, since an arithmetic register may not operate concurrently with a memory register. The other schemes attempt to preserve speed by splitting the memory so that a special register incorporated in one section may operate concurrently with an addressable register in the other section.

An additional selection mode is assumed by the Scheme I design but not by the others. In that mode, the word specification may be changed between bits, primarily with the purpose of activating the bits of two words alternately. If the two word registers are i and j, respectively, the output channels during reading would contain the sequence b_{i1}, b_{j1}, b_{i2}, b_{j2}, etc. The two registers could be chosen independently from the normally selected or the specially selected section. The remaining characteristics in this section apply to all schemes.

An important assumption on reading and writing is that a given bit may be changed immediately after it is read. If reading is destructive, then some data must be restored by a writing action; data is assumed to be restorable either directly or in modified form. If reading is nondestructive, special means are assumed for undelayed writing.

Finally, special equipment is assumed for input and output functions. Three functions must be served: the presentation of data to external loads, the storage of data derived from external sources, and the signalling of interrupt requests from external sources. It is assumed that all of these have the form of addressable registers, so that their contents may be read or recorded as if they were ordinary memory words. In this way, the various tasks of priority execution, format control, buffering, etc., may be accomplished by executing subroutines rather than by providing special logical equipment. The only logically special behavior required is isolation of the reading circuits, so that input data and control signals, which are not synchronous with internal computer operations, do not interfere with normal reading. This could be accomplished either by special gates on the reading circuits, which would have to be activated when the access switch selected the appropriate words, or by electrical techniques, which would limit the rate of change of flux during the receipt of external signals.

The following four subsections will describe several organization schemes based on the viewpoints above, and on the special memory properties just described. The first three will be sketched briefly, with emphasis on structure and on basic data processes. The fourth will be described in greater detail, in order to illustrate approximately how much the complexity of the hardware might be diminished and how much the complexity and slowness of calculation is increased with respect to a conventional organization.

E. SCHEME I–SINGLE MEMORY

1. STRUCTURE

In this scheme, only a single memory is used, with the only external registers being the two address registers of the memory itself and the execution control register (OP). As shown in Fig. 1.4, the data paths are very few in number, *i.e.*, a single path from memory to the single-bit arithmetic unit, and a path carrying either the unmodified or modified arithmetic unit output to the instruction register, the two access registers, and back to the memory itself. All memory data flows through the arithmetic unit.

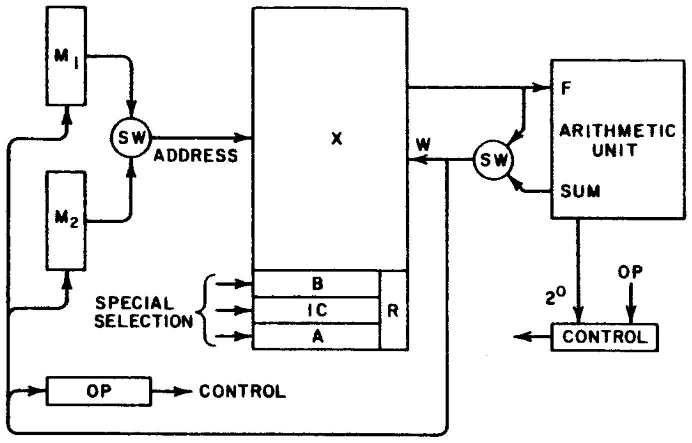
The memory is assumed to have the special behavior described in Subsection III-D, *i.e.*, the bits of two independently chosen words are selected alternately, both for reading and for writing. In this way, two words are processed at the same time in time multiplex. Two memory addressing registers are needed, not for this alternation of words, but because at a certain time, one register must specify a location whose contents will be read into the second register. The register outputs feed only the memory access drivers, making possible some electrical design simplifications. Three special registers are provided in the main memory, which may be addressed normally, or by direct action of the control, *i.e.*, the accumulator, A, the instruction counter, IC, and an index register, B.

2. BASIC MACHINE CYCLE

The cycle of operations has three stages, as follows:

a. Step

The contents of the special instruction counter register (IC) are read and incremented by 1, passing through the arithmetic unit. The new value is restored in IC, and recorded in memory access register M_2.



STEP	IC, M_1	$\leftarrow$	$IC + 1$
FETCH	M_2, OP	$\leftarrow$	$X_1 + B/0$
EXECUTE			
ARITHMETIC	R	$\leftarrow$	$\pm X_2 + R/0$
STORE	X_2	$\leftarrow$	$0 + R$
JUMP	IC	$\leftarrow$	X_2

A-3196-71

FIG. 1.4 BLOCK DIAGRAM, SCHEME I COMPUTER

b. Fetch

Register M_2 is used to obtain the next program instruction. As the first bit of the instruction is read, it is tested. If the bit is 1, the special index register, B, is activated, and the sum of the index and the instruction is formed and recorded in access register M_1. The reading and restoring to memory is multiplexed, as previously explained. If the first instruction bit read is 0, the special index register is not activated.

c. <u>Execute</u>

Three basic, alternative operations are provided: arithmetic, storage, and transfer of control. Only a brief explanation will be given here. A more detailed explanation will be presented in Scheme IV.

Arithmetic is limited to addition and subtraction, with or without pre-clearing of the accumulator. Symbolically, if A is the content of the accumulator and X the content of the selected operand register, the following alternatives are possible.

$A \leftarrow A + X$ In all cases, $X \leftarrow X$

$A \leftarrow 0 + X$

$A \leftarrow A - X$

$A \leftarrow 0 - X$

(The sign $\leftarrow$ signifies that the new value of the function at the arrow's head is obtained by performing the operations shown at the arrow's tail.)* The 0's are written in order to suggest the mechanism of logical control, that is, if the number 0 corresponds to the absence of a signal, a simple AND gate implements the optional choice.

The least significant bit appears first in time, due to the characteristics of 2's complement arithmetic, the sign bit—conveniently called the 2^0 bit—appears last, and no re-calculations are needed for any combinations of signs, *i.e.*, the sign is automatically correct if it is calculated as the most significant binary digit of the sum. Special storage is provided in the arithmetic unit to preserve the sign of an arithmetic operation during the EXECUTE phase, so that it may be tested in a subsequent transfer-of-control operation. This special storage is needed, since the same arithmetic unit is also used in the subsequent STEP stage.

Storage consists of the copying of the contents of the special accumulator register into the specified operand memory cell, *i.e.*,

$X \leftarrow A.$

In both storage and arithmetic, register M_1 specifies the location of the operand, and special control activates the accumulator register. A very useful variation, which will be described more fully in Scheme IV, is Double Precision, *i.e.*, operations on pairs of consecutive words. It is convenient to be able to treat a pair of words as a single word of double length, with the sign given by that of the most significant part—to be called the "head"—, and the sign of the less significant part—to be called the "tail"—suppressed. The selection of such pairs is

* The notation used here is inspired by the work of Iversen[8]

facilitated by restricting the locations of the "tail" words to, say, only the even memory locations. In this way, the location of the "head" is obtained simply by changing the least significant bit of the address register.

Transfer of Control, also to be called "Jump", is accomplished by copying the contents of the specified operand memory cell into the instruction counter, *IC*, register. A conditional transfer of control may be specified, in which the transfer depends upon the sign of the last arithmetic operation, as stored in the arithmetic unit.

If a return linkage to a program sequence is desired, such as in the use of subroutines, it must be accomplished by programming, *i.e.*, the contents of the *IC* register is copied into some reserved memory cell prior to the execution of the transfer of control.

3. SUMMARY

The scheme is very economical of equipment and data paths external to the memory. Some of the inconvenient features are

(1) the need for the special multiplex mode of memory operation, which is time-consuming, and

(2) no two of the three basic stages of the machine cycle may be concurrent.

The basic operations may be summarized symbolically as follows:

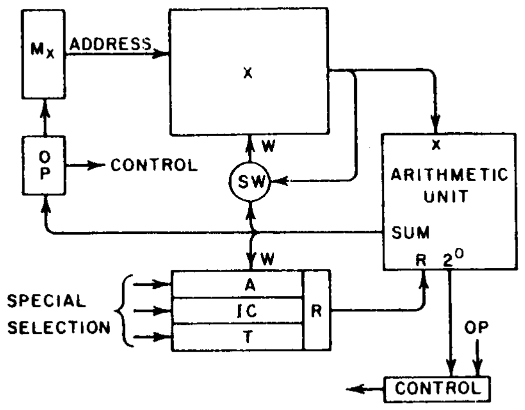
(1)	STEP	$IC, M_1 \leftarrow IC + 1$
(2)	FETCH	$OP, M_2 \leftarrow X_1 + B/O$
(3)	EXECUTE	
	arith	$R \leftarrow R/O \pm X_2$
	store	$X_2 \leftarrow R + O$
	jump	$IC \leftarrow X_2$

where R represents any of the special registers
$a, b \leftarrow$ means that both a and b are set to the same value
R/O means that either R or O may be chosen.

F. SCHEME II ONE MEMORY, EXTERNAL REGISTER BANK

1. STRUCTURE

The second scheme, whose block diagram is given in Fig. 1.5, illustrates a possible compromise between Scheme I and the hypothetical conventional organization. A bank of special registers is provided externally to the memory, but it is restricted to the operation of only one register at a time. The intention of the restriction is to allow the construction of the bank in the



STEP	IC, M_X	$\leftarrow$	IC + I
FETCH (a)	T	$\leftarrow$	X + 0
(b)	M_X	$\leftarrow$	T + 0
EXECUTE			
ARITHMETIC	R_i	$\leftarrow$	$\pm X + R_i / 0$
STORE	X	$\leftarrow$	$0 + R_i$
JUMP	IC	$\leftarrow$	0 + X

A-3196-72

FIG. 1.5 BLOCK DIAGRAM, SCHEME II COMPUTER

form of a memory, with a simple selection switch, and with a single read-write amplifier serving the entire set. It may reasonably be expected that common bit selection equipment may serve both the memory and the registers.

In Scheme I, two memory address registers were provided, so that one might control the selection of an instruction, while the other receives the operand address data of the selected instruction. In Scheme II, one of these registers, here labled T, is included in the register bank, so that only one independent memory address register is required. The price paid for this economy is the addition of a fourth stage in the basic machine cycle. The same arrangement is possible in Scheme I.

The external registers are an accumulator, A, the instruction counter, IC, and a transfer register T, all represented by the general symbol R. The arithmetic unit now receives its inputs simultaneously, since there are separate amplifiers for the memory and for the register bank. Three outputs are required from the arithmetic unit, two for direct restoration of data to the respective input sources, and the sum, which may be recorded in either source, and which also carries the data for the operation-code register OP and for the memory address register M_x.

2. SUMMARY OF OPERATIONS

Using the notation of the previous section, the four stages are:

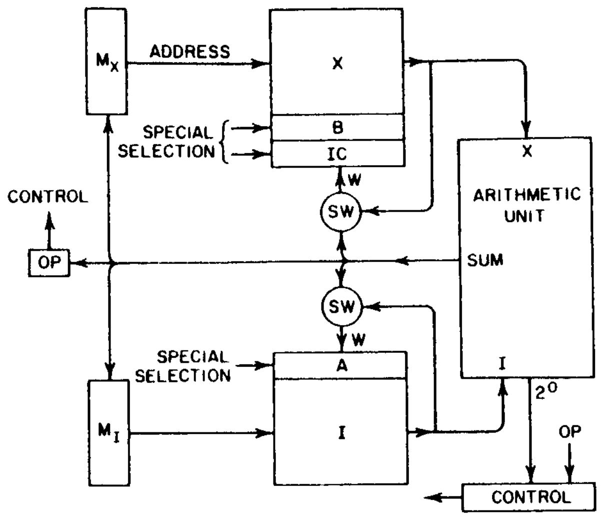
(1)	STEP	$M_x, IC \leftarrow IC + 1$
(2)	FETCH(a)	$T \leftarrow X + 0$
	(b)	$M_x \leftarrow T$
(3)	EXECUTE	
	arith	$R \leftarrow R/O \pm X$
	store	$X \leftarrow R + 0$
	jump	$IC \leftarrow X + 0$

G. SCHEME III–TWO MEMORIES

1. STRUCTURE

Scheme III (Fig. 1.6) carries forward the idea of a separate memory-like bank to its logical extension, *i.e.*, the use of a second full-fledged memory. Its data-path structure is completely symmetrical. It is convenient to assign instruction storage to one memory, here labeled I, and general operand storage to the other, labeled X, and to assign special registers to the memories, as in Scheme I. This division allows an instruction from the general I store to be read simultaneously with a special index register, B, in the X store, and an operand from the general X store to be read simultaneously with the accumulator, A, in the I store.

The arithmetic unit thus operates as in Scheme II. The sum output is used to send instruction address data to the lower memory register M_I, during the STEP stage, and the operand address and operation code data to the upper memory register M_x and the "op-code" register OP, respectively, during the FETCH stage. The basic operations are summarized in paragraph 2 below, and some special problems created by this structure are discussed in paragraph 3, below.



STEP	$M_I \leftarrow$ IC+1
FETCH	$M_X \leftarrow$ I +B/O
EXECUTE	
ARITHMETIC	A $\leftarrow$ ±X+A
STORE	X $\leftarrow$ O+A
JUMP	$M_I \leftarrow$ X

A-3196-73

FIG. 1.6 BLOCK DIAGRAM, SCHEME III COMPUTER

2. SUMMARY OF OPERATIONS

Three sequential stages are needed, as summarized below

(1)	STEP	$M_I, IC \leftarrow IC + 1$
(2)	FETCH	$M_x \leftarrow I + B/O$
(3)	EXECUTE	
	arith	$A \leftarrow A/O \pm X$
	store	$X \leftarrow A + O$
	jump	$M_I \leftarrow X$

3. SPECIAL PROBLEMS

There are three functions which are somewhat unusually affected by the splitting of operands and instructions into separate memories, *i.e.*, transfer of control, instruction modification, and memory loading.

As shown in 2, transfer of control (jump) is by a kind of indirect addressing, *i.e.*, the instruction specifies not the address of a new instruction, but the location of an X operand which contains that address. This requires extra memory space and some bookkeeping for the programmer (more likely, the compiler), but there are no logical difficulties. It also makes convenient certain kinds of instruction modification, *i.e.*, of the "set switch" type.

On the other hand, ordinary operations on instruction words are not convenient, inasmuch as the EXECUTE stage is designed to employ the upper memory. This is of no great consequence, inasmuch as the indirect jump and the availability of B-modification provide sufficient programming generality.

The unavailability of the I memory during regular EXECUTE time is, however, a problem for the actual loading of the I memory itself, either during the initial preparation for service, or in ground-controlled program correction. Several alternative solutions are available whose merits depend upon certain unknown factors.

It may be that the instruction set will never change, or that its changes may be accomplished via the indirect jump addressing feature. If the instruction set is in permanent-storage form, it might be built separately, and simply loaded as a physical package. For the more general case, two solutions, at least, are possible.

First, a special operation phase may be provided, to accept input data in the form of a Location-Information word pair, *i.e.*, the first word appearing at the input terminals is sent to M_I, which then specifies the location in I to which the second word is sent.

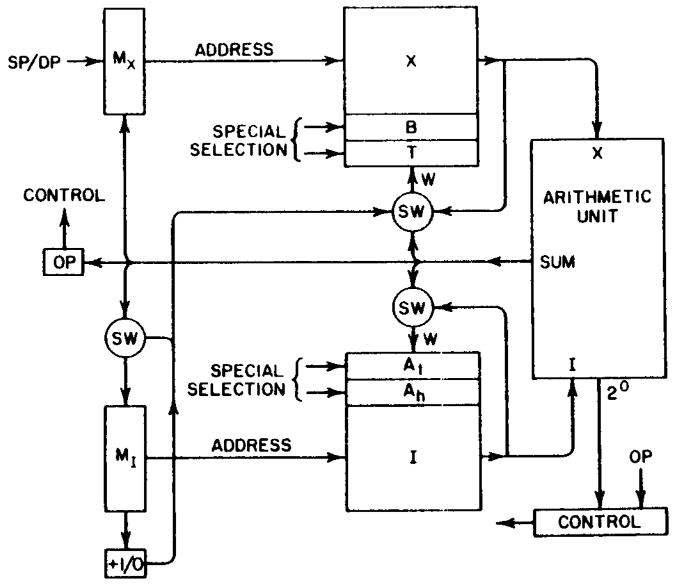
Second, the symmetry of the data structure may be exploited by providing for the programmed exchange of roles of the X and I memories. Thus, special instructions for loading I may be provided in the X memory, and the control may be altered so that instructions are taken from X and operands taken to and from I.

H. SCHEME IV—TWO MEMORIES, SPECIAL COUNTER

1. STRUCTURE

The fourth scheme whose block diagram is shown in Fig. 1.7, is the most elaborate of all. It is an extension of scheme III, in that separate memories are provided for instructions and operands, each with its own address register, and the special registers are divided between, and included within, the two memories. The additional feature is the provision for the independent stepping of the instruction address register M_I, by circulation through the equipment labeled "+1." The purpose of this feature is to save time by allowing the STEP and EXECUTE phases to be concurrent.

Several features necessary to the detailed operation of a machine are included here which were ignored in the previous descriptions for simplicity of explanation. These are



FETCH	M_X	$\leftarrow$	$I + B/0$
STEP	M_I	$\leftarrow$	$M_I + 1$
EXECUTE			
ARITHMETIC	A	$\leftarrow$	$\pm X + A$
STORE	X	$\leftarrow$	$0 + A$
JUMP	M_I	$\leftarrow$	X

A-3196-74

FIG. 1.7 BLOCK DIAGRAM, SCHEME IV COMPUTER

(1) equipment for double precision operations, and

(2) equipment to facilitate response to external interrupt commands.

The former is indicated by showing two Accumulator words, A_t and A_h, and by indicating that the operand memory address register M_x may be stepped up by changing its least significant bit at the end of the "tail" word. The response to interruption is facilitated by provision of a special register in the X memory, T, to receive the contents of M_I, which specifies the location of the current instruction, so that a return may be made to the normal instruction sequence after interruption.

2. SUMMARY OF OPERATIONS

The basic operations are the same as for Scheme III, but the separate counting facility enables the STEP and EXECUTE to occur simultaneously.

(1)	FETCH	$M_x \leftarrow I + B/O$	
(2)	STEP	$M_I \leftarrow M_{I+1}$	
	EXECUTE		
	arith	$A \leftarrow A/O \pm (X + 2^{-n}/O)$	Single or Double Precision
	store	$X \leftarrow A + O$	Single or Double Precision
	jump	$M_I \leftarrow X$	
	interrupt	$T \leftarrow M_I,\ M_I \leftarrow O$	

In arithmetic, the optional addition of the term 2^{-n} is included because it appears to be only trivially expensive to do so, due to the nature of 2's complement arithmetic.

The remarks on special problems caused by a split memory, made at the end of Subsection III-G, apply also to the present scheme.

3. INTERRUPT

Since no instructions take longer to execute than a double-precision addition, it is feasible to delay response to an interrupt command which occurs during an instruction until the next execution phase. At that time, the contents of M_I are transferred to the T register, and M_I is cleared to zero. The next instruction is taken from location zero in the normal way, but this will be the first step of a special interrupt subroutine. At the completion of the subroutine, a transfer-of-control is ordered, which uses T as its indirect address, in the normal manner, thus returning to the regular instruction sequence.

I. ARITHMETIC AND PROGRAMMING

This subsection will give examples of subroutines and programming techniques in order to help assess the reduction in computation efficiency resulting from the stringent reduction in control and arithmetic equipment.

1. ARITHMETIC FOR ADDITION, COMPLEMENTATION AND SUBTRACTION

a. Introduction

The rules of two's complement arithmetic will be summarized here (with apologies to most readers) in order to enable the reader to follow the multiplication subroutine to be described in paragraph III-H-3, below.

Let x be an n-bit number in the range $-1 \leq x < 1 - 2^{-n}$, and let x_r be its representation in the computer. For $x \geq 0$, $x_r \equiv x$, and for $x < 0$, $x_r \equiv 2 + x$. An immediate consequence of these definitions is that the leading digit is 0 for x positive, and 1 for x negative. Some interesting numbers are (for $n = 5$),

x	x_r
$1 - 2^{-n}$	0.11111
2^{-n}	0.00001
0	0.00000
-2^{-n}	1.11111
-1	1.00000

All arithmetic is done modulo 2, *i.e.*, carries from the 2^0 place are lost. The rules of arithmetic are as follows:

b. Addition and Subtraction

The representation of a sum or difference of two numbers is the sum or difference of their representations,

i.e., $(x \pm y)_r = x_r \pm y_r$.

This may be accomplished by performing ordinary binary addition or subtraction on all the bits of x and y, including the 2^0 bit, ignoring carries from the 2^0 position.

c. Two's Complementation

The radix complement of x, *i.e.*, $2 - x$, is obtained by adding 2^{-n} to the one's complement of x. The latter is obtained simply by logically complementing the bits of x. That is, for $n = 5$,

$$(2 - x)_r = 2 - x_r = (2 - 2^{-n}) - x_r + 2^{-n}$$

$$= (1.11111 - x_r) + 2^{-n}$$

$$= (\text{bit complement of } x_r) + 2^{-n}.$$

The 2^{-n} term is easily added in a serial arithmetic unit by presetting the carry flip-flop prior to the entry of the least significant bit.

d. Mechanization of Subtraction

It is convenient to mechanize subtraction as the sum of the minuend and the two's complement of the subtrahend, *i.e.*, $(y - x)_r = y_r + (2 - x)_r$

$$= y_r + (\text{bit complement of } x_r) + 2^{-n}.$$

If the 2^{-n} term is not entered, the result is

$$y - x - 2^{-n}.$$

e. Multiplication by Two

Doubling may be done by adding a number to itself. This is also called left-shifting (by one place).

f. Double Precision

In multiplication it is convenient to expand a number of n bits into a $2n$ bit format, such that if the original, single precision number is x, the double precision number is $2^{-n}x$. The new number will be contained in two computer words, with the more and less significant parts called the "head" and "tail," respectively.*

For $x \geq 0$, $x_r\,(DP) = 2^{-n}\,x_r\,(SP)$,

e.g., for $x_r\,(SP) = 0.\text{abcde}$,

$x_r\,(DP) = 0.00000\text{abcde}$

For $x < 0$ $x_r\,(DP) = 2 + 2^{-n}\,x(SP)$, (note: x here, not x_r)

e.g., for $x_r\,(SP) = 1.\text{efghi}$

$x_r\,(DP) = 1.11111\text{efghi}.$

* This terminology, and the scheme for multiplication are taken from Ref. 3

g. Summary

The arithemetic unit of the proposed computers will be restricted to the performance of the operations described above, utilizing a special register and a memory register as possible inputs and outputs. Considerable flexibility of operation is available if facilities are provided for letting each input and output terminal be connected independently to either of the two associated registers, or to neither (for inputs, this is equivalent to the reading of zero). Thus the possible output may be summarized in the following formula,

$$R/X \leftarrow R/X/0 \pm (R/X/0 + 2^{-n}/0),$$

where R represents the special register, X the memory register, $\pm$ represents an optional sum or difference, / represents independent choice of associated registers, and n is the number of bits in the fractional part of the single- or double-length word.

2. MNEMONICS FOR INSTRUCTIONS

Several subroutines will be given in succeeding paragraphs. This paragraph will provide some names and mnemonics for the machine operations, for use in presenting the subroutines.

Table 1.3 below specifies the basic operations and their variations, separately. The operations will be described by reference to a basic operation

$$z \leftarrow t/0 \pm (u + 2^{-n}/0)$$

where z, t, and u will be specified as being the contents of A, the accumulator, or X, the selected memory register. Some, but not necessarily all, of the alternative choices, signified by / and $\pm$, will be specified by the various elementary operations. Numerical examples will be given for $n = 5$.

Table 1.4 lists the combinations of the basic functions and their variations which will be employed later. The mnemonic for the combined operation is obtained by combining the mnemonic symbols for the component operations.

3. MULTIPLICATION

In the computers described, multiplication is accomplished as a subroutine. The details of a suitable program, written in terms of the basic operations listed in the previous paragraph, are given in Appendix A to Part One. Since the program is to be used as a subroutine, it would appear only once in memory, and its internal structure would be ignored in ordinary program-writing or compiling.

TABLE 1.3

ELEMENTARY FUNCTIONS

Function	Mnemonic	Operation
Addition	A	$A \leftarrow N_o + (X + 2^{-n}/0)$
Subtraction	S	$A \leftarrow N_o - (X + 2^{-n}/0)$
Shift	Sh	$z \leftarrow z + (z + 2^{-n}/0), \quad z = A$ or X
Store	St	$X \leftarrow A$
Jump	J	Copy X into the Instruction Counter
Jump if sign was +	J_o	Copy X into the Instruction Counter
<u>Variations</u>		
Increment	I	$z \leftarrow t/0 \pm (u + 2^{-n})$
Clear (accum.)	Cl	$z \leftarrow 0 \pm (u + 2^{-n}/0)$
Double precision	D	$n' = 2n$

TABLE 1.4

BASIC OPERATIONS

Operation Class	Single Precision		Double Precision	
	No Increment	Increment	No Increment	Increment
Add, clearing A	AC	ACI	ACD	ACDI
Add to A	A	AI	AD	ADI
Subtract, Clearing A	SC	–	SCD	–
Subtract from A	S	–	SD	–
Shift (left one place)	Sh	ShI	ShD	ShDI
Store	St	–	StD	–
Jump if sign was 0	J_o	–	–	–
Jump (unconditionally)	J	–	–	–

The program consists of two parts, a preface, and the main iteration. The preface program occupies 14 word spaces and has an average length of 10.5 instructions, one of which is in double precision. The loop occupies seven words in memory, and takes an average of 5 1/2 instruction times, one of which is in double precision. The average time for execution of a multiplication subroutine for an N bit number is given by the following expressions:

Preface:	$9.5\, t_s + t_d$
Loop:	$(4.5\, t_s + t_d)N$,

where t_s and t_d are the times taken for single and double precision instructions, respectively. The instruction cycles for the schemes described require a number of stages, each of which involves the serial processing of the bits of one or two words. Scheme I requires three or four word times, each involving a two-phase clock, for a total of $6N$ or $8N$ clock periods. Scheme II requires four or five words, each with single phase clocking, for a total of $4N$ or $5N$ periods. Scheme III requires $3N$ or $4N$ periods, and Scheme IV requires $2N$ or $3N$ periods.

The total multiplication times for $N = 20$ for the various schemes are:

Scheme I	$35N^2 + 65N$	15,300 clock periods
Scheme II	$23N^2 + 43N$	10,060 clock periods
Scheme III	$17.5N^2 + 32.5N$	7,650 clock periods
Scheme IV	$12N^2 + 22N$	5,240 clock periods

For comparison, we may assume that the hypothetical ordinary computer would do multiplication in a two-stage process, the first being the fetching of instruction, requiring N clock periods, and the second being the execution, requiring N^2 clock periods, for additions and N for shifting, *e.g.*, for $N = 20$, the multiplication time is 440 clock periods (this assumes that all registers operate simultaneously, and are capable of right-shifting).

The elimination of the multiplication command thus has resulted in a slow-down of multiplication by a factor.of from 12 to 36 times, compared to the hypothetical, ideal serial computer.

Two circumstances may mitigate this slowness. First, since the product is developed from the most significant digits downward, it is possible to truncate the subroutine after only a fraction of the N steps, according to the precision desired. Second, multiplication by simple constants might be programmed as an unconditional sequence of shifts and additions.

In Paragraph III-J-5 below (Variations on the Design) it is shown that the multiplication might be speeded significantly by an increase of control equipment (by about two flip-flops and associated logic), without changing the basic data path structure and timing, *e.g.*, for Scheme II, multiplication would be eight times rather than 23 times slower than the standard hypothetical computer. The remaining slowness results from the fact that only one register may operate at a time (concurrently with the memory) and that only left shifting is possible.

In absolute terms, if a conservative memory cycle time of 20 μsec is assumed with a resulting bit rate of 50 kc, the fastest multiplication time (without the modification mentioned in the preceding paragraph) would range from about 10 msec to 300 msec. This appears to be adequately fast for the computations required.

An interesting figure for the comparison of different schemes would be the total energy cost of a multiplication. The four schemes with variants described, take many times more clocks than a typical hypothetical computer, but fewer components operate at each clock pulse. The computation of a valid estimate of power consumption is beyond the scope of this report.

4. DIVISION

There are a number of feasible division algorithms. The one chosen for illustration here is the "restoring" type, in which all numbers are processed in absolute magnitude form. The subroutine starts with a preface, in which negative numbers are complemented and the sign of the quotient is determined. Upon determination of the sign, a program "switch" is set which will or will not complement the quotient after its calculation in the main loop.

The details of the program are given in Appendix A. The average preface operation, plus the final switch, takes 15 steps, and the average loop takes 7 steps, all single precision. The loop is traversed N times. The total numbers of clock periods required for the four schemes for $N = 20$ are:

Scheme 1	$42N^2 + 90N$	18,600 clock periods
Scheme 2	$28N^2 + 60N$	12,400 clock periods
Scheme 3	$21N^2 + 45N$	9,300 clock periods
Scheme 4	$14N^2 + 30N$	6,200 clock periods.

This scheme is probably the least efficient of the common algorithms. It takes about 20% more time than multiplication, but it usually occurs much less frequently.

J. LOGICAL DESIGN

The design equations for several component blocks will be presented here in order to illustrate the simplicity of the logical equipment of the various schemes. A comprehensive design is not considered to be justified for the purpose of this report.

1. ARITHMETIC UNIT

The following inputs are required:

Data: Augend A $\quad -1 \leq A, B \leq 1 - 2^{-n}$
Addend B

Control: "Clear accumulator"	k_{CL}
"Complement B"	k_{CO}
"Increment"	k_{IN}
"EXECUTE Phase"	p_{EX}

Timing pulses: begin word (precedes first bit)	t_b
end word (at last bit)	t_e

The arithmetic operation performed is

$$S - A/0 \pm (B + 2^{-n}/0),$$

where S is the sum output, delivered to the "accumulator." Also, the value of S at the 2^0 bit time (the last, most significant bit) during the execute phase, is stored in a flip-flop, D. For clearing the accumulator, $k_{CL} = 1$, for addition, $k_{CO} = 0$, for subtraction, $k_{CO} = 1$ and $k_{IN} = 1$. If $k_{IN} = 1$ during addition, the sum is moved by increments of 2^{-n}, and if $k_{IN} = 0$ during subtraction, the difference is moved by decrements of 2^{-n}. The control signals are obtained from the operation code bits directly or by decoding. All must be present throughout the full operation, except for k_{IN}, which is needed only for one bit time at the beginning of the operation.

The logical equations of the arithmetic unit are the following, where C is the carry flip-flop and D is the sign storage flip-flop:

$$\text{Set } C = k_{IN} t_b + A(B \oplus k_{CO})\bar{k}_{CL}$$
$$\text{Reset } C = \bar{k}_{IN} t_b + \bar{A}\,(\bar{B} \oplus k_{CO})$$

$$S = C \oplus A\,\bar{k}_{CL} \oplus B \oplus k_{CO}$$

$$\text{Set } D = S\,t_e\,k_{EX}$$

$$\text{Reset } D = t_b\,k_{EX}.$$

Logical operations may be accomplished with only slight additions. For example, the function $A\ B$ may be mechanized by augmenting S as follows:

$$S = (C \oplus A\,\bar{k}_{CL} \oplus B \oplus k_{CD})\,(k_{LA} + A).$$

The new control signal, k_{LA}, is normally 1; for the logical AND operation, $k_{LA} = 0$, $k_{CO} = 0$, and $k_{CL} = 1$. The last condition causes C to be 0, so the S is reduced to $(0 \oplus 0 \oplus B \oplus 0)\ (0 + A)$, or AB. The logical complement of B bits is available in the original equations, when $k_{IN} = 0$, and $k_{CL} = k_{CO} = 1$.

2. "PLUS – ONE" UNIT.

The circuit used for shifting by increments the instruction location register in Scheme IV requires one carry flip-flop, C. The inputs are:

Register Data	R	
Control	C_p, where $C_p = 1$ for	$R \leftarrow R + 1$
	$C_p = 0$ for	$R \leftarrow R + 0$
Timing	t_b, a pulse occurring prior to the first bit.	

The equations for flip-flop C and for the output sum are:

$$\text{Set } C = t_b C_p$$

$$\text{Reset } C = \overline{R}$$

$$(\text{Sum})R' = R \oplus C$$

3. OPERATION CODE

The following operation code design is appropriate to Scheme II. It assumes the availability of eight bits in order to minimize decoding. Three bits, k_{CL}, k_{CO} and k_{IN}, control the arithmetic unit independently, and one bit, k_{JC}, distinguishes between conditioned and unconditioned transfer of control. Three bits k_{R1}, k_{R2}, and k_{R3}, together, control selection of source and distinction regisiers for arithmetic operations; included implicitly among these combinations are the store and transfer of control ("jump") functions.

The functions of the control bits will be described relative to a basic EXECUTE operation:

$z \leftarrow t + u + v$ (arithmetic sum)

with	Z =	the selected load of z
	T =	the selected source of t
	U =	the selected source of u
	v =	2^{-n} or 0
	A =	Accumulator
	IC =	Instruction Counter
	X =	Memory

Bit Name	Symbol	Effect of variable	
		$k = 0$	$k = 1$
Clearing	k_{CL}	$t = T$	$t = 0$
Complement	k_{CO}	$u = U$	$u = 1 - U$
Increment	k_{IN}	$v = 0$	$v = 2^{-n}$
Jump Condition	k_{JC}	conditional	unconditional
Double Precision	k_{DP}	single	double

Register selection

Function	k_{R1}	k_{R2}	k_{R3}	Z	T	U
Accumulate	0	1	1	A	A	X
Shift Accum	0	1	0	A	A	A
Jump ($X_{CL} = 1$)	0	0	1	IC	0	X
Store Accum	1	1	0	X	A	0
Shift mem	1	0	1	X	X	X
Store IC	1	0	0	X	IC	0

Note: $(k_{R1} = 1) \leftrightarrow (Z = X)$
$(k_{R2} = 1) \leftrightarrow (T = A)$
$(k_{R3} = 1) \leftrightarrow (U = X)$

4. CONTROL LOGIC

This paragraph will list the design equations for a control section appropriate to Scheme II. The design is not expected to be without flaws but it gives a fair idea of the simplicity and small size of the control equipment. It includes all operations listed in the previous section, and it also includes interrupt response.

The control section receives inputs from a timing generator and from the bits of the op-code register. It contains four flip-flops and produces 15 output signals. The logical equations for all of these are as follows:

INPUTS:

Basic Timing

t_b a pulse occurring in a vacant cell between words.

t_e a pulse occurring with the most significant (2^0) bit.

OP code register bits*

$q_{CL}, q_{CO}, q_{IN}, q_{JC}, q_{DP}, q_{R1}, q_{R2}, q_{R3}$

Arithmetic Unit Sign Storage Flip-Flop

D ($D = 0$ for positive sum)

CONTROL FLIP-FLOPS

cycle phase	r_1:	Trigger $r_1 = T_{r1} = t_b\,(\overline{p}_4 + \overline{k}_{DP})$
	r_2:	Trigger $r_2 = T_{r2} = T_{r1}\, r_1$
double precision	k_{DP}:	set $k_{DP} = p_4 t_b q_{DP}\, \overline{k}_{DP}$
		reset $k_{DP} = p_1 + t_b\, k_{DP}$
interrupt	k_{IR}:	set $k_{IR} =$ (external interrupt)$\cdot\ \overline{p}_4$
		reset $k_{IR} = p_4 t_b$

CONTROL SIGNALS (Internal & External)

The cycle phase flip-flops define the four basic machine cycle phases:

STEP:	$p_1 = \overline{r}_1\, \overline{r}_2$
FETCH(a):	$p_2 = \overline{r}_1\, r_2$
FETCH(b)	$p_3 = r_1\, \overline{r}_2$
EXECUTE	$p_4 = r_1\, r_2 = p_{EX}$

* If timing pulses were available to distinguish the bits of the op-code as they are read from memory, three op-code register bits could be eliminated, as follows:

q_{IN} could set the arithmetic unit Carry flip-flop directly

q_{JC} could set up the data path control flip-flops k_{R1}, k_{R2}, and k_{R3} subject to the state of the sign storage flip-flop D, and

q_{DP} could set the double precision flip-flop k_{DP} directly.

Elementary control signals are obtained directly—or with slight modification—from the *OP*-code register:

$$\begin{aligned} k_{Cb} &= p_2 + q_{CL} & \qquad k_{DP} &= q_{DP} \\ k_{CO} &= q_{CO} & k_{R1} &= q_{R1} + k_{IR} \\ k_{IN} &= p_1 + q_{IN} & k_{R2} &= q_{R2}\,\bar{k}_{IR} \\ k_{JC} &= q_{JC} & k_{R3} &= q_{R3}\,\bar{k}_{IR} \end{aligned}$$

The above are used externally, at the arithmetic unit, and internally in the control unit.

The following signals control the reading and writing of the memory and the register bank.

Clear $OP\text{-}M_x$ = $p_4 k_{IR}$

Load $OP\text{-}M_x$ = $\bar{r}_2$

Read X = r_2

X Write Control = $p_2 + p_4\,(k_{R1} + \bar{k}_{R2} + k_{R3})$

X Write Data = $p_2 X' + p_4\,(k_{R1} S + \bar{k}_{R2} X')$

R write control = $\bar{r}_1 + \bar{r}_2 + q_{JC} + k_{R1} + k_{R2} + \bar{k}_{R3}$

R write Data = $(\bar{r}_1 + r_1 r_2 \bar{k}_{IR}) S + r_1 \bar{r}_2\, R'$

Select A_h = $p_4\,\bar{k}_{DP}\,k_{R2}$

Select A_t = $p_4\,k_{DP}\,k_{R2}$

Select IC = $p_1 + p_4\,(\bar{k}_{R1}\,\bar{k}_{R2}\,k_{R3}(q_{JC} + \bar{D}) + k_{R1}\,\bar{k}_{R2}\,\bar{k}_{R3})$

Select T = $p_2 + p_3$

Note on Interrupt Action

An external "interrupt" signal is stored in a special flip-flop, k_{IR}, and is ignored until the following Execute phase. At that time, the memory address register M_x is cleared, so that the location 0000 is selected, the content of the *IC* register is stored there, and the *IC* register itself also is cleared to zero. The machine then proceeds into a normal Step phase, at which time, the *IC* content (0000) is stepped to (0001). The location 0001 will provide the first step of a subroutine, which puts away the content of *A*, performs the necessary Interrupt functions, and then returns to the regular program, by transferring control to the contents of cell 0000, *i.e.*, the original instruction location.*

*Inasmuch as the transfer of control will be followed by the movement by increments of the new *IC*, and the instruction specified by the *IC* was interrupted, the subroutine must decrement the contents of cell 0000 by 1 so that the instruction not be skipped.

5. VARIATIONS ON THE DESIGN

Given a design—whose characteristics depend heavily upon the intuition of the designer—it is natural to consider changes in efficiency with increases or decreases in equipment. As it stands, the design of the previous section requires four flip-flops in the control, two in the arithmetic unit, and eight in the op-code register. The latter could be reduced to five, with additional decoding.

One attractive simplification is elimination of the double precision feature. The net saving would be about one flip-flop, with its set/reset logic. The major cost would be a lengthening of the multiplication loop by the addition of a sign test and the extra STEP and FETCH stages of another addition instruction. For Scheme II, the multiplication time changes from $23N^2 + 43N$ to $30N^2 + 43N$, or, for $N = 20$, from 10,060 to 12,860 clock periods. This represents about a 28% time increase for a saving of one flip-flop. The elimination of double precision also reduces the length of the op-code register from a range of 8-5 to 7-4, thus there would be a flip-flop saving of from 7% to 18%.

The major target for time reduction is the multiplication loop. Several possibilities exist. One simple scheme is merely to mechanize only the multiplication loop given in Figure 6. There are four distinct processes, requiring two new control flip-flops (it is assumed that the tests require only combinational logic). It is also necessary to add a register to the register bank to store the multiplier (the reliability loading of this register should be fairly low—only a word of cores is added to an existing driver-amplifier set). The time savings in this scheme would be appreciable, *i.e.*, only $6.5N^2 + 3N$ clocks are needed instead of $23N^2$. For $N = 20$, 6540 clocks are saved out of a total loop time of 9200, for a net multiplication time of 3520 clocks instead of an original 10,060. Thus, for an increase in logical equipment of about 25%, the multiplication time is reduced by a factor of about 3. Such an exchange is probably justified, since the time saved by the increased efficiency could well be used, *e.g.*, in checking calculations.

IV SUMMARY AND CONCLUSIONS

Most of this part has been devoted to an investigation of the feasibility of the application of certain design emphases in computer system design, *i.e.*, the severe restriction of the amount of control and arithmetic equipment and the number of separate, independently active, data paths, at the price of increased memory size, decreased programming convenience, and reduced computations and, perhaps, increased power consumption. Several sample designs have been presented which carry out these emphases to varying degrees. None of these has been absolutely minimal, since it was desired to preserve a reasonable computation power. The schemes appear capable of achieving the necessary computation speeds using conservative memory speeds.

Compared to conventional computers, the computation power per machine element is low. Questions of "efficiency" or engineering esthetics, however, should be subordinated to the need to maximize reliability within the confines of the computation speed required for the given task. A proper evaluation is admittedly difficult, because, for example, it might be argued—in the design phase—that any extra computation power beyond the minimum might be put to use in such reliability-serving tasks as double-checking, test exercises, etc.

A second topic considered has been the desirability of giving special protection to operations on the program data. It has been shown possible to build a computer in which the only operation on program data was that of transfer between registers, thus affording the application of powerful error-correction techniques. The computation speed of the sample scheme is low, and the memory capacity required is high, but the use of fixed-memory storage, which is usually inexpensive and compact, eases the memory capacity problem somewhat. The ultimate feasibility of the idea has not been evaluated.

V REFERENCES FOR PART ONE

1 Kautz, W. H., "Codes and Coding Circuitry for Automatic Error Correction Within Digital Systems," Tech. Report 2, SRI Project 3196, Stanford Research Institute, Menlo Park, California (January 1962).

2 Frankel, S.P., "On the Minimum Logical Complexity Required for a General Purpose Computer," *IRE Trans. EC-7(4),* p.283 , (December 1958).

3 Van der Poel, W. L., "A Simple Electronic Computer," *Applied Scientific Research Sect. B2,* p. 367 (1951).

4 Alonso, R., and Laning Jr., J. H., "Design Principles for a General Control Computer," MIT Instrumentation Laboratory Report R276, AD 242 861, Massachusetts Institute of Technology, Cambridge, Massachusetts, (April 1960).

5 Hunter, L.P., and Bauer, E.W., "High-Speed Coincident-Flux Magnetic Storage Principles," *J. Appl. Phys. 27(11),* p. 1257, (November 1956).

6 Lawrence Jr., W.W., "Recent Developments in Very-High-Speed Magnetic Storage Techniques," *Proc. Eastern Joint Computer Conference,* (December 1956).

7 Rosin, R.F., "List Structures and their Implementation through Advanced Machine Design," Research Report RC-297, IBM Research Center, Yorktown Heights, New York (August 1960).

8 Iversen, K.I., *A Programming Language,* (John Wiley and Sons, Inc., New York, N.Y., 1962).

APPENDIX A

MULTIPLICATION AND DIVISION SUBROUTINES

APPENDIX A
MULTIPLICATION AND DIVISION SUBROUTINES

I. MULTIPLICATION

The multiplication subroutine must be accomplished in two stages: the iteration, whose rules are the same as in absolute-value multiplication, and the preface, which performs a certain correction necessitated by the peculiarities of the two's complement notation. In addition, the preface in the routine to be described also converts the factors into double precision form, to achieve the results normally accomplished by the two accumulator registers in conventional arithmetic units.

A. PREFACE

1. Format

Six memory registers are prepared. The first two are used for the head and tail of the multiplicand, expanded into double precision format as described in Section I above. The second two are the head and tail Special Accumulator Registers, used for developing the product. The fifth is used for the multiplier, which is used directly, both for positive and negative numbers. The sixth is used to keep a running account of the number of interations. It is set to 0.00001 initially, and shifted for each iteration, so that the arrival of a 1 at its sign position signals the end of the multiplication.

2. Correction

A negative multiplier x is represented as the binary number $2- |x| = 2 + x = 1 + (1 + x) = 1.(1 + x)$. Following the formal rules for multiplication, with a multiplicand y, the part on the right side of the binary point produces the nominal product $y(1 + x)$. The proper value, yx, may be had by subtracting the value y from the nominal product.

Thus, for negative multipliers, the multiplicand must be subtracted from the accumulated product. This may be done either before or after the regular multiplication. It is most conveniently done together with double precision formating.

The preface operation is summarized in Fig. A-1. The program occupies 14 word spaces, and has an average length of 10.5 instructions, one of which is in double precision.

Two possible modifications of this preface are

1) Rounding–This may be done by entering an initial 2^{-n} in the accumulator.

2) Variable precision–This may be done by setting the tally register to any desired initial starting point.

3. The Regular Loop

In the regular multiplication loop, the double-precision multiplicand is accumulated into the double-precision accumulator subject to the control of the multiplicand bits, as in integer multiplication. Since only a left-shift is possible, the multiplier bits are scanned (by testing the bit in the sign position) most-significant-digit first. This requires the accumulator to be shifted leftward, while the multiplicand remains stationary, entering into successively lesser places of the accumulator. For each test of a multiplier bit, the tally register is shifted, and the arrival of 1 at the sign place indicates the end of the loop.

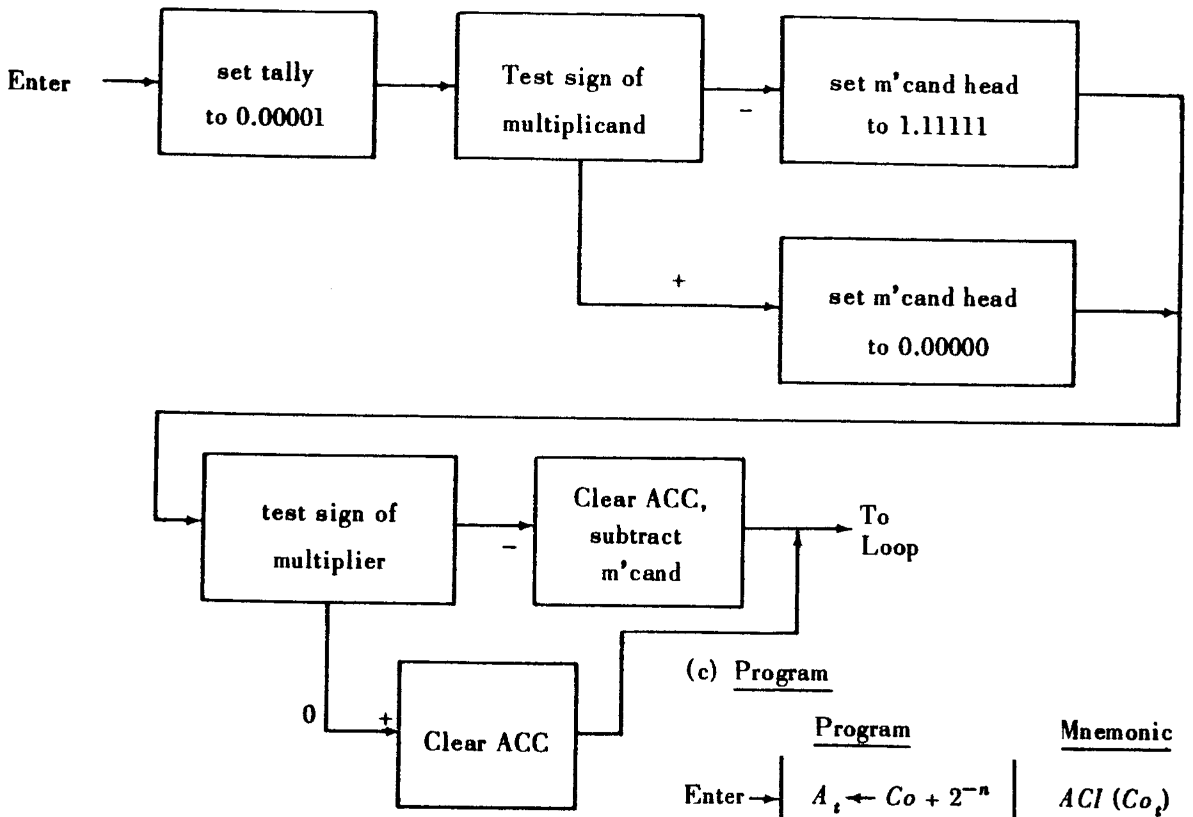
A flow diagram and program chart are shown in Fig. A-2. The address for the return transfer to the main program is presumed to be set up by the program prior to entry to the subroutine.

B. DIVISION

Division also is accomplished by a preface and a loop program. The purpose of the preface is to convert all numbers to absolute value form and to provide for the complementation of the Quotient, if needed. The loop carries out the familiar "pencil-and-paper" or "restoring" division algorithm.

Flow diagrams are given in Figs. A-3 and A-4.

(a) Flow Diagram



Enter
set tally to 0.00001
Test sign of multiplicand
–
set m'cand head to 1.11111
+
set m'cand head to 0.00000
test sign of multiplier
–
Clear ACC, subtract m'cand
To Loop
0
+
Clear ACC

(c) Program

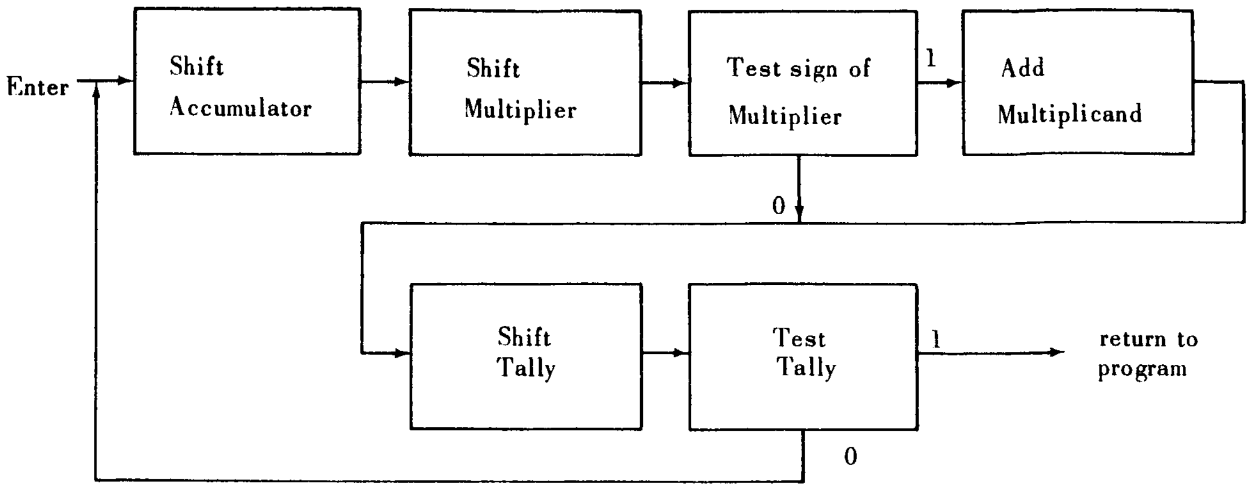
	Program	Mnemonic
Enter →	$A_t \leftarrow Co + 2^{-n}$	$ACI\ (Co_t)$
	$T \leftarrow A_t$	$St\ (T)$
	$A_t \leftarrow y_t$	$AC\ (y_t)$
	$A \geq 0$	J_o
	$A_t \leftarrow Co - 2^{-n}$	$C_mC\ (Co_t)$
	J	J
	$A_t \leftarrow Co$	$AC\ (Co_t)$
	$y_h \leftarrow A_t$	$St\ (y_h)$
	$A_t \leftarrow x$	$AC\ (A_t)$
	$A \geq 0$	J_o
	$A_{h,t} \leftarrow -y_{h,t}$	$SCD\ (y_{h,t})$
exit ←	J (Loop)	J
	$A_{h,t} \leftarrow Co_{h,t}$	$ACD\ (Co_{h,t})$
exit ←	J (Loop)	J

(b) Register List

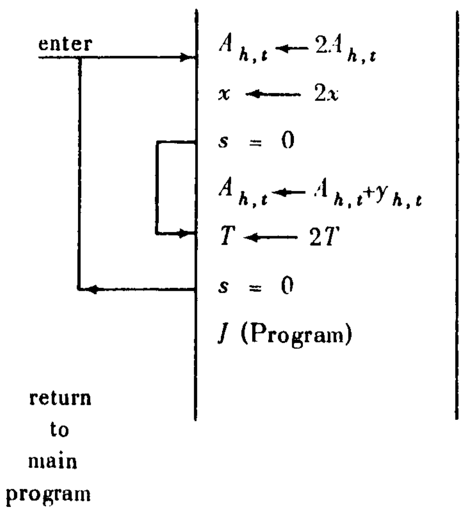
Register Name	Symbols
Multiplicand	y_h, y_t
Multiplier	x
Tally	T
Accumulator	A_h, A_t
constant (zero)	Co_h, Co_t

FIG. A-1 MULTIPLICATION SUBROUTINE (I – Preface)

(a) Flow Diagram



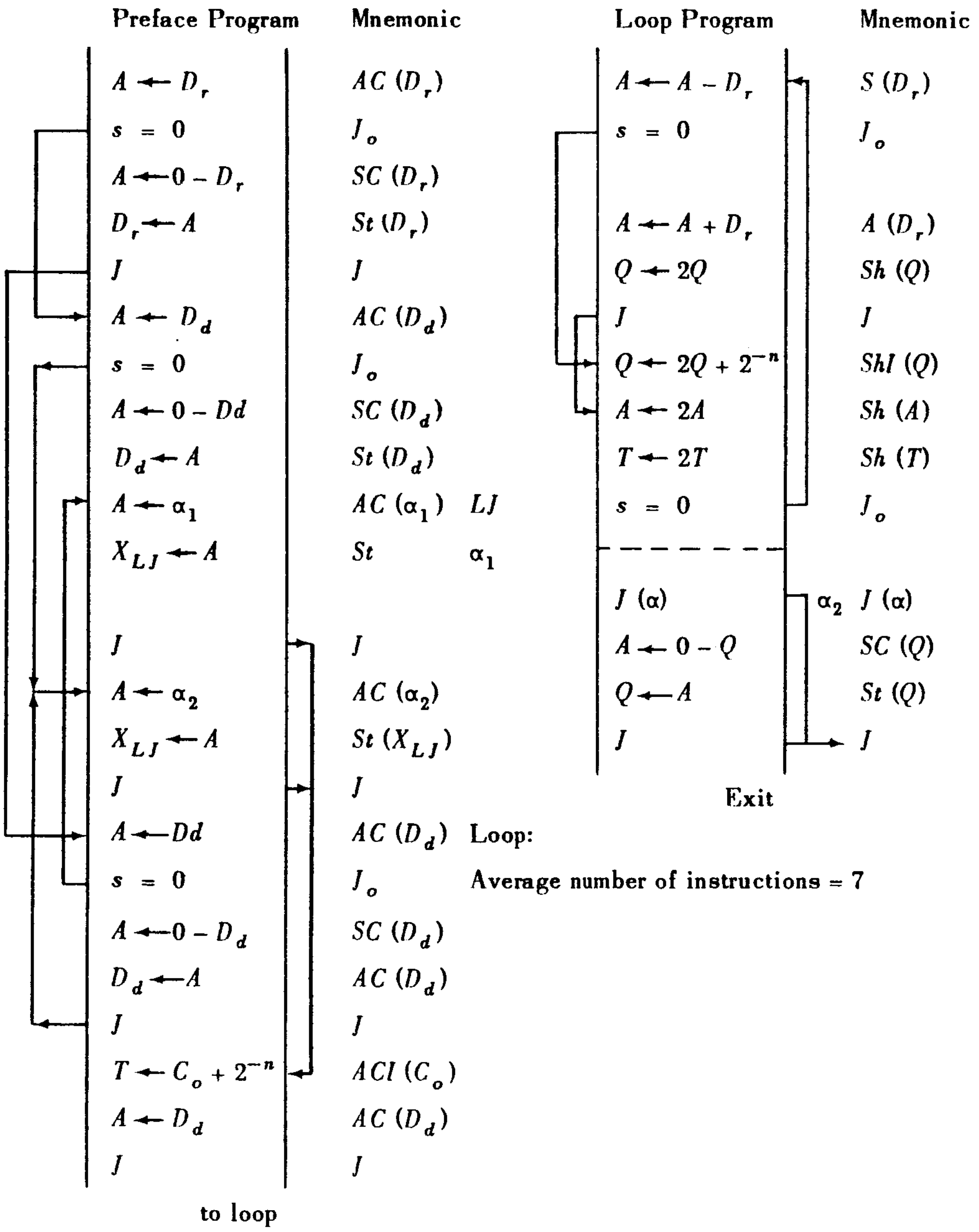
(b) Program



Operation	Mnemonic
$A_{h,t} \leftarrow 2A_{h,t}$	$ShD\ (A_{h,t})$
$x \longleftarrow 2x$	$Sh\ (x)$
$s = 0$	J_o
$A_{h,t} \leftarrow A_{h,t}+y_{h,t}$	$AD\ (y_{h,t})$
$T \longleftarrow 2T$	$Sh\ (T)$
$s = 0$	J_o
J (Program)	J

Note: *s* represents the sign memory FF. It records the sign of the last arithmetic operation.

FIG. A-2 MULTIPLICATION SUBROUTINE (II – Main Loop)



Preface Program	Mnemonic	
$A \leftarrow D_r$	$AC\ (D_r)$	
$s = 0$	J_o	
$A \leftarrow 0 - D_r$	$SC\ (D_r)$	
$D_r \leftarrow A$	$St\ (D_r)$	
J	J	
$A \leftarrow D_d$	$AC\ (D_d)$	
$s = 0$	J_o	
$A \leftarrow 0 - Dd$	$SC\ (D_d)$	
$D_d \leftarrow A$	$St\ (D_d)$	
$A \leftarrow \alpha_1$	$AC\ (\alpha_1)$	LJ
$X_{LJ} \leftarrow A$	St	α_1
J	J	
$A \leftarrow \alpha_2$	$AC\ (\alpha_2)$	
$X_{LJ} \leftarrow A$	$St\ (X_{LJ})$	
J	J	
$A \leftarrow Dd$	$AC\ (D_d)$	Loop:
$s = 0$	J_o	Average number of instructions = 7
$A \leftarrow 0 - D_d$	$SC\ (D_d)$	
$D_d \leftarrow A$	$AC\ (D_d)$	
J	J	
$T \leftarrow C_o + 2^{-n}$	$ACI\ (C_o)$	
$A \leftarrow D_d$	$AC\ (D_d)$	
J	J	

to loop

Loop Program		Mnemonic
$A \leftarrow A - D_r$		$S\ (D_r)$
$s = 0$		J_o
$A \leftarrow A + D_r$		$A\ (D_r)$
$Q \leftarrow 2Q$		$Sh\ (Q)$
J		J
$Q \leftarrow 2Q + 2^{-n}$		$ShI\ (Q)$
$A \leftarrow 2A$		$Sh\ (A)$
$T \leftarrow 2T$		$Sh\ (T)$
$s = 0$		J_o
$J\ (\alpha)$	α_2	$J\ (\alpha)$
$A \leftarrow 0 - Q$		$SC\ (Q)$
$Q \leftarrow A$		$St\ (Q)$
J		J

Exit

D_r	=	Divisor
D_d	=	Dividend
Q	=	Quotient
C_o	=	Constant (zero)
α_1	=	address of complementing program exit
α_2	=	address of non-complementing program exit

FIG. A-3 DIVISION SUBROUTINE PROGRAM

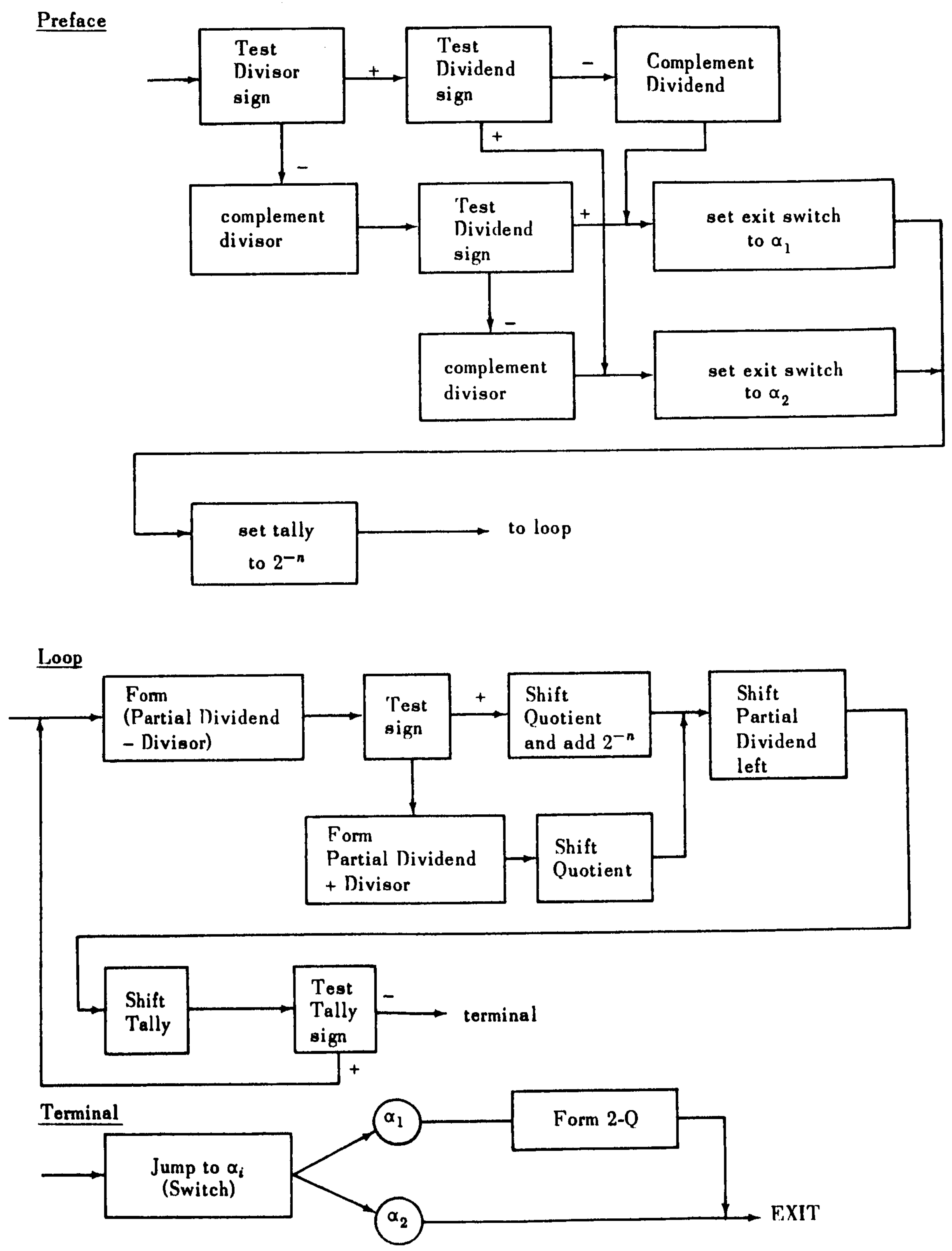


FIG. A-4 DIVISION SUBROUTINE (Preface and Loop)

PART TWO

RELIABLE MEMORY TECHNIQUES*

I INTRODUCTION

In Part Three of this report, a number of reliable computer organizations are studied. A common feature of all these computers is the assumption of one or more very reliable memories. For, if some of the various arithmetic and other registers are contained in memory, the results of a memory failure can be disastrous indeed. On the other hand, suppose the memory can be made reliable by using techniques such as those covered in the present part of this report. Then highly reliable computers which are essentially all memory (so-called All-Memory Computers) might be possible.

Of the various organizations which are available for memory, perhaps arrays of rectangular hysteresis loop magnetic cores are the most versatile. Unlike magnetic drums, they involve no moving parts; unlike acoustic delay lines, they require essentially no standby power. Furthermore, magnetic core memories (and aperture plate memories as well) have been improved over the past decade to a considerable degree of reliability. Therefore, only magnetic core memories will be treated in this part of the report.

It is of interest now to consider the various portions of a magnetic core memory. There are first the memory cells. If an initially-reset memory cell is energized with more than some threshold magnetomotive force, its magnetization changes state. If it receives less than this threshold magnetomotive force there is essentially no permanent change in magnetization. One may speak of a *selection ratio*, p, for the array of memory cells. This is the ratio of the least magnetomotive force that is intended to change the state of a memory cell to the greatest magnetomotive force that is not intended to change the state. As memory cells are not ideal devices, and as the currents which determine the applied magnetomotive forces are not entirely constant, it is desirable from a reliability standpoint to operate with as large a selection ratio as possible.

* by Robert C. Minnick

There are two essentially different ways in which the memory cells may be arrayed: *word-organization* and *bit-organization.* A simple word-organized memory consisting of four three-bit words is shown as Fig. 2.1. In this type memory, each word is threaded by a separate *read-access* wire, R_i. The reading of a word consists of applying a current to one of the read-access wires; as long as this current exceeds the threshold of the memory cells, the reading of information in the memory cells is properly accomplished. It is useful to extend the definition of selection ratio, which was previously given: The *read selection ratio,* p_R, is the selection ratio for the memory cells during the read operation, and the *write selection ratio,* p_W, similarly is the selection ratio for the memory cells during the write operation. Thus for a word-organized memory, such as the one shown in Fig. 2.1, $p_R = \infty$. This means that if the maximum read current variation is known for a word-organized memory, the nominal operating current always may be selected so as to reduce the probability of failure to any predetermined value.

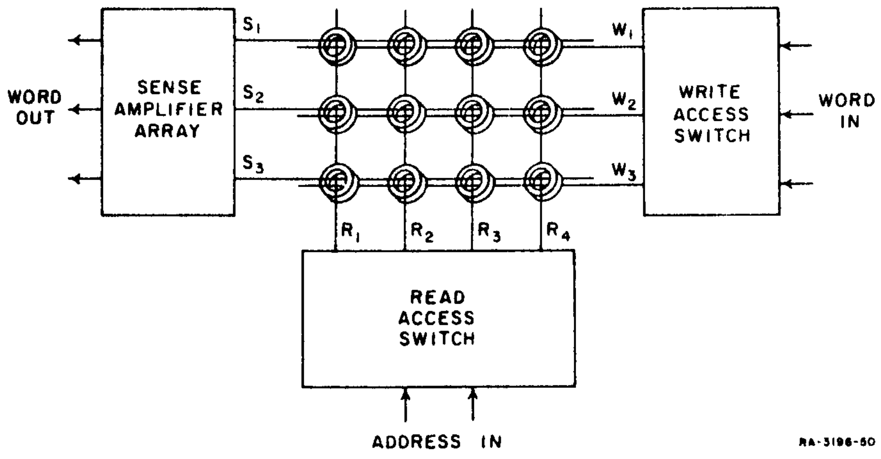


FIG. 2.1 WORD-ORGANIZED MEMORY SYSTEM

Referring again to Fig. 2.1, writing is accomplished in a word-organized memory by passing a current through one of the wires, R_i, simultaneously with currents or no currents on each of the wires, W_i. It is clear that the best possible write selection ratio for this memory is $p_W = 3$, although normally such a system is operated with $p_W = 2$; therefore, there exists a more severe restriction on the variations in the currents for the write operation than for the read. This observation is true in general.

A small bit-organized memory is shown as Fig. 2.2; it consists of four three-bit words. To read a word in this memory one wire of each of the x and y read access switches is energized. To write, the same R_{xi} and R_{yj} wires are energized together with appropriate wires W_k. The best possible selection ratios for bit-organized memories as characterized by Fig. 2.2 are $p_R = 3$, $p_W = 2$; although normally such a system is operated with $p_R = 2$, $p_W = 2$. Bit-organized memories normally require even more careful control of the read and write currents than do word-organized memories. If the memory cells are multiapertured, it is possible[1*] to arrange the system so that significant variations of the read and write currents may occur without upsetting proper operation.

Techniques have been developed for increasing p_R and p_W[2,3,4,5]; these techniques consist in increasing the number of read selection wires that thread each core over the one or two which are shown in Figs. 2.1 and 2.2. By using such techniques, the memory cells need not have such uniform properties, and the read and write selecting currents need not be as carefully controlled. Methods for increasing the read and write selection ratios will not be further considered in the present report.

Figures 2.1 and 2.2 show several components of magnetic core memories other than the memory cells. These include the read access switch(es), the write access switch and the sense amplifier array. For a word-organized memory there is only one read access switch; for a bit-organized memory there are two. In either case, this portion of the system completely decodes an address and supplies current to a single output wire. Such switches must deliver a constant output current on the proper wire under a number of possible failure conditions. The design of these switches will be considered in Sec. II. The write access switch receives an input code word, and under a number of possible failure conditions it must supply a constant output current on the proper output lines. The design of these switches will be considered in Sec. III.

The sense amplifier array serves to detect and amplify the weak signals emitted by the memory cells during the read operation. This array must be arranged such that it properly distinguishes ZEROS from ONES on the sense lines S_i for given ranges of permissible signals. Furthermore, it should operate properly under some conditions of electronic failure. An obvious approach towards making sense amplifier arrays reliable is the use of circuit redundancy[6]. The design of these arrays will not be considered further in this report.

---

* **References are grouped at the end of Part Two.**

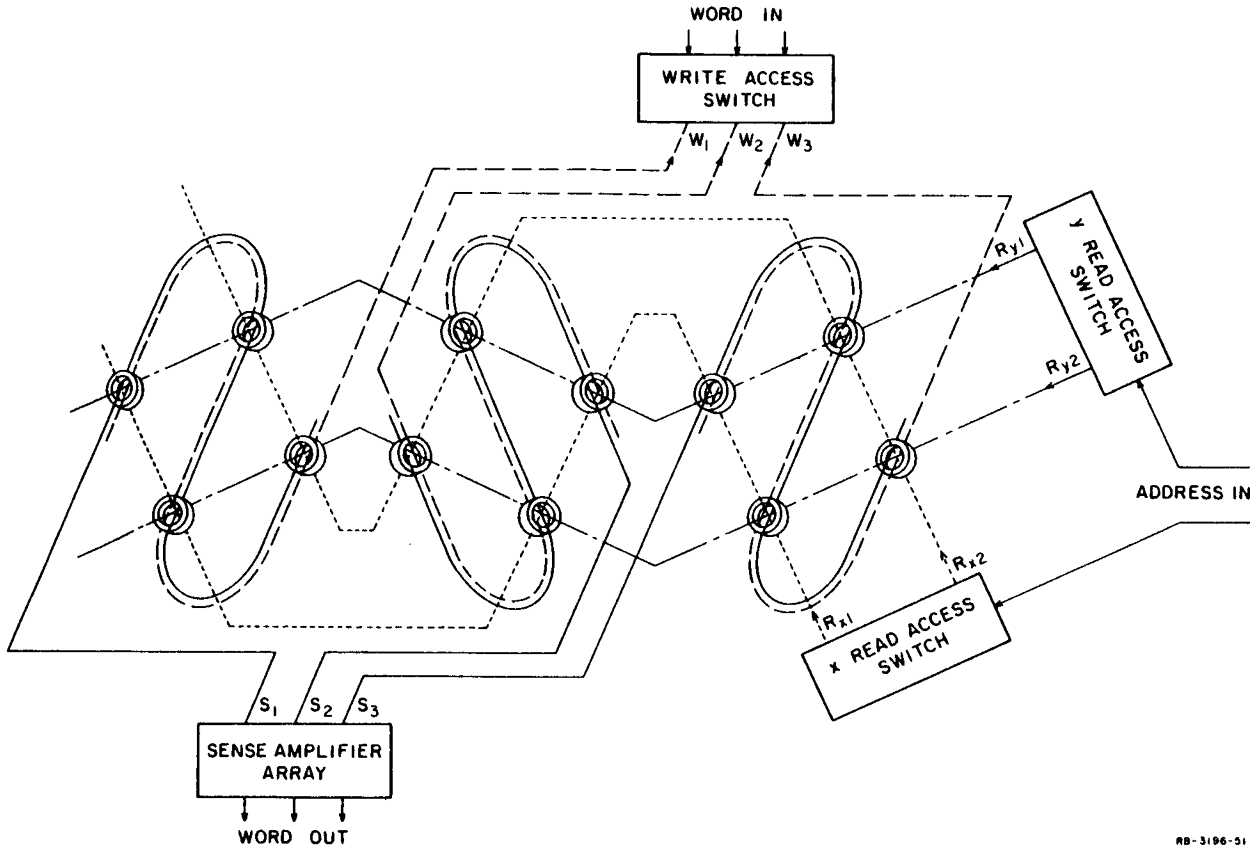


FIG. 2.2 BIT-ORGANIZED MEMORY SYSTEM

In the following sections the assumption will be made that improper operation in magnetic core memories is much more likely to be the result of semiconductor failures than of magnetic failures. This assumption, of course, is to some extent inaccurate; for the properties of magnetic cores do change, and wires do develop shorts. However, the evidence seems to indicate that a reasonably large percentage of failures after the initial checkout are due to semiconductor failures. Therefore, the approach that will be taken is to complicate the magnetics in such a way as to reduce the dependence on properly-operating semiconductors.

II READ ACCESS SWITCHES

A. DEFINITIONS

The design of reliable read access switches will be considered first. Note that the read access switch might consist of as many semiconductor drivers as there are R_i or R_{xi} and R_{yj} lines (see Figs. 2.1 and 2.2). However, by using a magnetic array together with one (or perhaps two) drivers for each address input wire, the total number of semiconductors in general can be significantly reduced. Therefore, it is of some interest to develop the characteristics of magnetic read access switches. As such a development is available[7], only an outline of the definitions follows.

An access switch is considered to have M input wires and M drivers, exclusive of the bias wire. The number of output wires is N. For example, the switch of Fig. 2.3 has $M = 6$ and $N = 9$. Each of the N output wires is associated with a unique core, and the M input wires in this switch are organized into n sets of m_i wires each, with $i = 1, 2, \cdots, n$, such that exactly one of the m_i wires in each of the n sets (n wires in all) is energized in the operation of the switch. For the switch in Fig. 2.3, Driver 1, 2, or 3 is energized simultaneously with Driver 4, 5, or 6. Therefore, there are two sets of three wires each, or $n = 2$, $m_1 = 3$, $m_2 = 3$.

There are a total of $M = \sum_{i=1}^{n} m_i$ input wires in the switch. Furthermore, any one of m_1 wires in the first set may be energized simultaneously with any one of m_2 wires in the second set, and so on. This means that there are a total number of $\pi_{i=1}^{n} m_i$ possibilities for energizing the M wires. For each of these possibilities, a different output is selected. Therefore, $N = \pi_{i=1}^{n} m_i$.

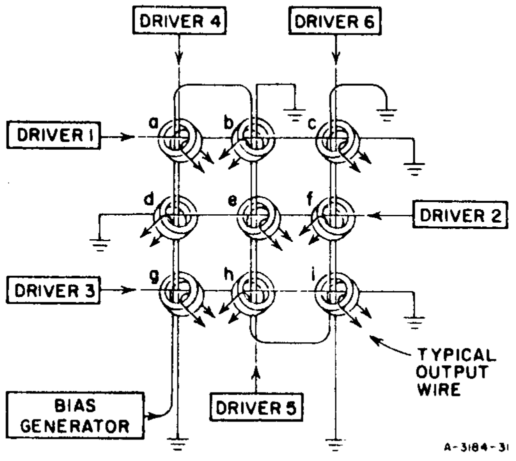
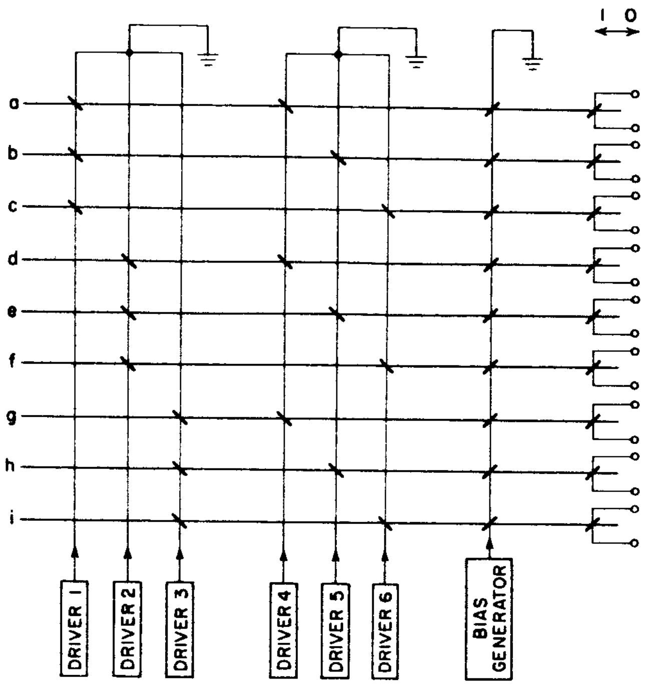


FIG. 2.3 READ ACCESS SWITCH FOR $n = 2, m_1 = 3, m_2 = 3$

An equivalent way to picture the read access switch of Fig. 2.3 is shown in Fig. 2.4. In this illustration the *mirror* symbolism is used, in which horizontal lines refer to cores and short diagonal lines refer to winding polarities. The remaining lines indicate connecting wires. With this mirror notation one may readily determine the contribution made by a winding to the magnetomotive force of a given core by reflecting the current on that wire onto the diagonal winding symbols associated with the core in question.

The switch drawn as Fig. 2.3 may be thought of as a geometric picture, because each of the $n = 2$ sets of $m = 3$ input wires is assigned a geometric dimension; therefore the parameter n is termed the *dimension* of the switch. One may consider a still more abstract picture of a switch, which has the advantage of allowing certain mathematical manipulations. In this form the switch is drawn as a two-dimensional array of integers. These integers represent the windings associated with each input wire on every core. For instance, considering the previous example, let a matrix of integers be written in a positional correspondence with the mirror symbols of Fig. 2.4 where +1 represents a positively-connected winding, −1 represents a negatively-connected winding, and 0 indicates no winding. This matrix, which is termed a *winding matrix*, becomes for the present example



NOTE: ALL WINDINGS ARE ONE TURN.

A-3184-32

FIG. 2.4 THE READ ACCESS SWITCH OF FIG. 2.3 IN MIRROR NOTATION

$$\underline{W} = \left[\begin{array}{cccccc|c}
1 & 0 & 0 & 1 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 & 1 & 0 & -1 \\
1 & 0 & 0 & 0 & 0 & 1 & -1 \\
0 & 1 & 0 & 1 & 0 & 0 & -1 \\
0 & 1 & 0 & 0 & 1 & 0 & -1 \\
0 & 1 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 1 & 1 & 0 & 0 & -1 \\
0 & 0 & 1 & 0 & 1 & 0 & -1 \\
0 & 0 & 1 & 0 & 0 & 1 & -1
\end{array}\right] \qquad (2.1)$$

In general a winding matrix is an $N \times M$ matrix, a row of which indicates the windings on one core, while a column indicates the connections which one winding makes to all cores. While this matrix representation loses the pictorial properties shown in Fig. 2.3, it retains sufficient information to construct the switch, without cluttering the picture with a maze of detail, much of which is irrelevant to the organizational properties. The winding to the right of the partition in Eq. (2.1) is the *bias winding.*

In a similar way, it is possible to define an $M \times N$ *selection matrix,* $\underline{S}$, a row of which indicates the currents delivered by one of the M drivers for all possible ways of turning on the drivers, and a column represents the currents produced by all drivers for one of the N possible ways of turning them on. It is easily seen that the $N \times N$ product matrix, $\underline{X} = \underline{W}\,\underline{S}$, which is termed the *excitation matrix,* contains elements x_{ij} which represent the resulting magnetomotive force applied to core i by energizing the drivers according to the *j-th* way of turning them on. Therefore, the elements x_{ii} along the major diagonal of $\underline{X}$ indicate the magnetomotive force received by the selected core, while the remaining elements indicate the magnetomotive force received by the unselected cores. If the bias is chosen so that the most positive off-diagonal elements of $\underline{X}$ are zero units of magnetomotive force, the value of the elements on the major diagonal (all of which can be shown to have the same value) is defined as the *load-sharing factor, d.* An equivalent and somewhat more intuitive definition of d is the ratio of the power delivered by the switch to the power supplied by *one* of the drivers. If $d > 1$, an appropriate choice of the bias can be made to assure proper operation, even though certain drivers fail in specified ways, as will be shown later in this section.

The winding matrix illustrated by the example of Eq. (2.1) may be written in a modified form. It is recalled that there are n sets of m_i columns (disregarding the bias winding). The structure of $\underline{W}$ is such that on any row there is exactly one ONE in any set of m_i columns. Then let a *code matrix,* $\underline{R}$, be defined to have $N = \prod_{i=1}^{n} m_i$ rows and n columns. For each row of $\underline{R}$, a digit 0, 1, . . ., $m_i - 1$ is placed in each of the n columns in accordance with the position of the single ONE in the corresponding set of m_i columns of $\underline{W}$. For example the $\underline{R}$ matrix corresponding to Eq. (2.1) is

$$\underline{R} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 2 & 0 \\ 2 & 1 \\ 2 & 2 \end{bmatrix} \qquad (2.2)$$

While the winding matrix of Eq. (2.1) was derived from the access switch of Figs. 2.3 and 2.4, such a matrix may refer to either of two fundamental types of switches: the *input logic* type or the *output logic* type. The input logic type is, in Olsen's[9] terminology, a *Type I switch.* It is the usual magnetic switch (illustrated in Figs. 2.3 and 2.4) for which each output is the secondary winding on a separate core, while each input is coupled to a series string of primary windings on several cores. This is called an input logic switch because the logic is performed in the primary windings of the transformers. The output logic type, or Olsen's *Type II switch,* is a switch for which each input is associated with a single primary winding on one core, while each output is coupled to a series string of secondary windings on several cores. This switch requires the use of additional threshold devices (usually diodes) in series with the output windings, because the cores are used as linear transformers and the logic is performed on the output windings. In either case, the same winding matrix may be used; only the meaning of the rows and columns of $\underline{W}$ is different. Therefore, in many of the considerations to follow, one may ignore the switch being an input logic or an output logic type. Where there is doubt, the input logic type is assumed.*

An access switch may be augmented by the addition of *redundant dimensions.* It is recalled that in the basic square switch there are n sets of m_i input wires each, such that one wire in each set is energized. These n sets are called *coordinate sets,* and it is recalled that the switch is said to be n-dimensional. Suppose that an additional s sets of m input wires each (where m is the largest of m_i) are added to the switch without changing the number of cores (and consequently outputs). It is possible to show that these additional sets, which are termed *redundant sets,* may be threaded such that if one driver in each redundant set is energized together with one in each coordinate set, the magnetomotive force received by the selected core is

* The redundant read access switches which have been developed by Takahashi and Goto[10] are of the *output* logic type.

enhanced, while the most positive magnetomotive force received by any non-selected core is not increased. One way of adding redundancy is by the use of *Latin n-cubes* having m elements on a side. Such a Latin n-cube is an n-dimensional cube with m symbols which are arranged so that along any coordinate line each of the m symbols occurs exactly once.

The addition of redundancy to an access switch allows the designer to achieve a great deal of flexibility. For instance, he may have available drivers which deliver only a certain amount of power, but he may need more than this amount of power delivered to the load. By adding redundancy, the load may be shared among two or more drivers in order to achieve this result. Furthermore, the inclusion of redundancy results in a switch which is somewhat less dependent on the various circuit tolerances than is the corresponding non-redundant switch. That is, the use of a sufficient amount of redundancy may allow one or more of the drivers to fail in certain ways without adversely affecting the over-all operation.

Suppose a non-redundant input logic switch having parameters n (dimensions) and $m_i = m$ (inputs per dimension) is compared with an output logic switch having the same parameters. It can easily be shown that both switches have mn drivers and m^n outputs. The input logic switch has m^n cores and n windings per core (exclusive of bias), while the output logic switch has mn cores and m^{n-1}windings per core. Thus, by changing from an input logic switch to an output logic switch, one may reduce the number of cores at the expense of increasing the number of windings per core and adding semiconductor diodes. The addition of redundancy to either type of switch increases the number of drivers over the corresponding non-redundant switch.

It will be convenient for later developments to define several parameters:

b is the bias magnetomotive force applied to each core,

e_i is the number of windings on each core if an input-logic type switch is constructed,

e_o is the number of windings on each core if an output-logic type switch is chosen, and

D is the number of diodes used in an input recoding network, if one is necessary.

B. SINGLE PARITY-CHECK READ ACCESS SWITCHES

Suppose the code external to the memory system is any block[11] code with a single parity check. Stuart-Williams[12] observed a number of years ago that if this code is also binary, an additional pair of drivers and the corresponding windings may always be added to incorporate this redundancy in the switch. For example, a switch of this type which corresponds to a three-bit code plus even parity has the winding matrix

$$\underline{W} = \left[\begin{array}{cccccc|cc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}\right] \qquad (2.3)$$

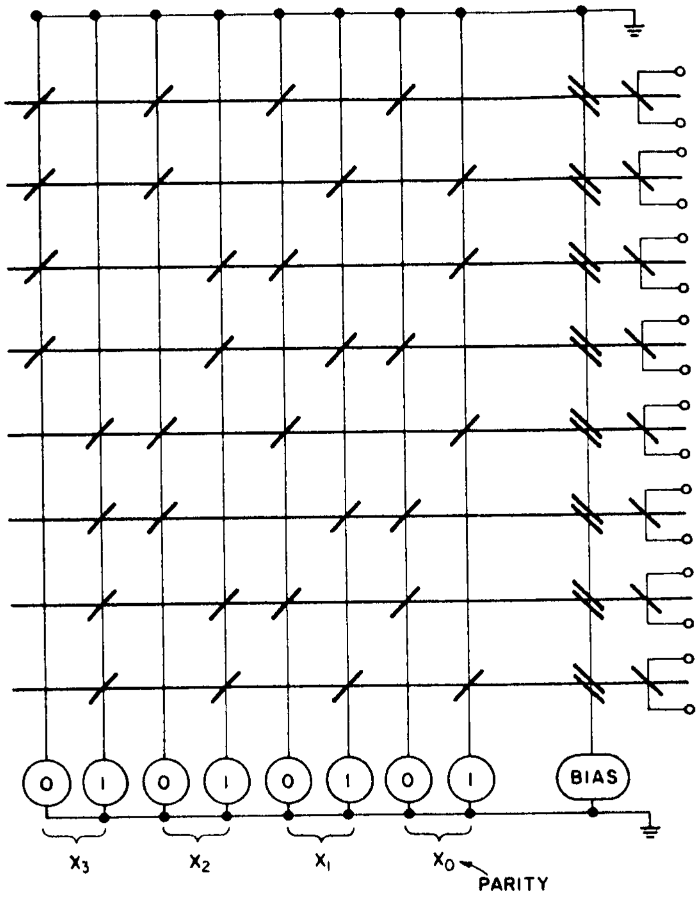
An input logic switch that corresponds to Eq (2.3) is shown as Fig. 2.5(a). For this switch each core is a horizontal line and each wire is a vertical line; for the output logic type these would be interchanged. Assuming the input code is error-free, the selected core is energized with a load-sharing factor of $d = 2$. The *distance** of this code, not accidentally, is also two. The added windings, which correspond to the parity column of the code, are nothing more than the redundant set of windings which correspond to the 2x2x2 Latin cube shown as Fig. 2.5(b).

The $\underline{R}$ matrix for this example is

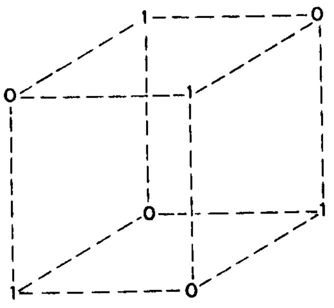
$$\underline{R} = \left[\begin{array}{ccc|c}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1
\end{array}\right] . \qquad (2.4)$$

It is seen that the $\underline{R}$ matrix contains on each row the various single parity-check codes which can occur at the input of the switch. This observation suggests an approach that might be used for a general development of single parity-check switches: start with an $\underline{R}$ matrix of the single parity-check codes, and from this construct the corresponding $\underline{W}$ matrix.

* The distance of a code is defined here in the Hamming[9] sense. That is, if every pair of radix q code words is compared and if d_i is the number of digits which disagree in the ith such comparison, then the distance d of the code is the minimum of the d_i.



(a)



(b)

RA-3196-53

FIG. 2.5 READ ACCESS SWITCH WITH PARITY WINDINGS

Suppose then for such a general development that one starts with a dense n-bit, q-radix code, and adds a parity-check column. The $\underline{R}_n$ matrix for this is

$$\underline{R}_n = \begin{array}{c|ccccccc|c|}
 & \multicolumn{7}{c}{\longleftarrow\; n \;\longrightarrow} & \\
\cline{2-9}
\uparrow & 0 & 0 & - & - & - & 0 & 0 & 0 \\
 & 0 & 0 & - & - & - & 0 & 1 & q-1 \\
q & & & - & - & - & & & \\
\downarrow & 0 & 0 & - & - & - & 0 & q-1 & 1 \\
\cline{2-9}
 & 0 & 0 & - & - & - & 1 & 0 & q-1 \\
 & 0 & 0 & - & - & - & 1 & 1 & q-2 \\
 & & & - & - & - & & & \\
 & 0 & 0 & - & - & - & 1 & q-1 & 0 \\
\cline{2-9}
 & & & - & - & - & & & \\
\cline{2-9}
 & 0 & q-1 & - & - & - & q-1 & 0 & n-2 \\
 & 0 & q-1 & - & - & - & q-1 & 1 & n-3 \\
 & & & - & - & - & & & \\
 & 0 & q-1 & - & - & - & q-1 & q-1 & n-1 \\
\cline{2-9}
 & 1 & 0 & - & - & - & 0 & 0 & q-1 \\
 & 1 & 0 & - & - & - & 0 & 1 & q-2 \\
 & & & - & - & - & & & \\
 & 1 & 0 & - & - & - & 0 & q-1 & 0 \\
\cline{2-9}
 & & & - & - & - & & & \\
\cline{2-9}
 & q-1 & q-1 & - & - & - & q-1 & 0 & n-1 \\
 & q-1 & q-1 & - & - & - & q-1 & 1 & n-2 \\
 & & & - & - & - & & & \\
 & q-1 & q-1 & - & - & - & q-1 & q-1 & n \\
\cline{2-9}
\end{array} \qquad (2.5)$$

The radix, q, is conveniently chosen to be a power of a prime, and the arithmetic must be appropriate to the radix[11, 13].

The first n columns of the code matrix of Eq. (2.5) may be written more compactly by noting that $\underline{R}_{n-1}$ occurs q times in $\underline{R}_n$. Therefore,

$$\underline{R}'_n = \begin{array}{cc} & \xleftrightarrow{\quad n-1 \quad} \\ q\updownarrow & \left[\begin{array}{c|c} 0 & \\ \cdots & \underline{R}'_{n-1} \\ 0 & \\ \hline 1 & \\ \cdots & \underline{R}'_{n-1} \\ 1 & \\ \hline & - - - \\ \hline q-1 & \\ \cdots & \underline{R}'_{n-1} \\ q-1 & \end{array}\right] \end{array} \tag{2.6}$$

The winding matrix which corresponds to this code matrix is

$$\underline{W}'_n = \begin{array}{cc} & \xleftrightarrow{\quad q \quad} \\ q\updownarrow & \left[\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & \\ 1 & 0 & 0 & 0 & 0 & \underline{W}'_{n-1} \\ & & \cdots & & & \\ 1 & 0 & 0 & 0 & 0 & \\ \hline 0 & 1 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & 0 & \underline{W}'_{n-1} \\ & & \cdots & & & \\ 0 & 1 & 0 & 0 & 0 & \\ \hline & & \cdots & & & \cdots \\ \hline 0 & 0 & 0 & 0 & 1 & \\ 0 & 0 & 0 & 0 & 1 & \underline{W}'_{n-1} \\ & & \cdots & & & \\ 0 & 0 & 0 & 0 & 1 & \end{array}\right] \end{array} \tag{2.7}$$

Thus, starting with the $\underline{R}_n$ matrix for a single parity-check code, Eq. (2.5), the corresponding winding matrix for the non-redundant portion of the read access switch has been derived, Eq. (2.7). The matrix $\underline{W}'_n$ has been written in the form shown because Eq. (2.7) has elsewhere (Ref. 2.6, Eq. (B-3)) been shown to be the general construction equation for a winding matrix. The complete winding matrix, $\underline{W}_n$, is obtained from $\underline{W}'_n$ of Eq. (2.7) by appending q columns in accordance with the rightmost column of Eq. (2.5). Examples of $\underline{R}$ and $\underline{W}$ matrices for several values of the parameters (q, n) are given as Fig. 2.6.

C. READ ACCESS SWITCHES HAVING DISTANCES GREATER THAN TWO

All of the codes (and read access switches) considered in the previous subsection are of a distance two, and all the switches are found to have a load-sharing factor of two. Each of the single parity-check read access switches will operate properly, but with a reduced output power, if any one driver fails by open-circuiting. If, on the other hand, the input code is in error by only one bit or if two drivers open-circuit, the single parity-check switch may operate improperly.

An obvious generalization to allow for more reliable operation is to develop read access switches which correspond to codes which have a larger distance. For instance, consider the switch that would correspond to the $d = 3$, seven-bit binary Hamming code. The $\underline{R}$ and $\underline{W}$ matrices for this code are

$$\underline{R} = \left[\begin{array}{cccc|ccc}
0&0&0&0&0&0&0\\
0&0&0&1&1&1&1\\
0&0&1&0&0&1&1\\
0&0&1&1&1&0&0\\
0&1&0&0&1&0&1\\
0&1&0&1&0&1&0\\
0&1&1&0&1&1&0\\
0&1&1&1&0&0&1\\
1&0&0&0&1&1&0\\
1&0&0&1&0&0&1\\
1&0&1&0&1&0&1\\
1&0&1&1&0&1&0\\
1&1&0&0&0&1&1\\
1&1&0&1&1&0&0\\
1&1&1&0&0&0&0\\
1&1&1&1&1&1&1
\end{array}\right], \qquad
\underline{W} = \left[\begin{array}{cccccccc|cccccc}
1&0&1&0&1&0&1&0&1&0&1&0&1&0\\
1&0&1&0&1&0&0&1&0&1&0&1&0&1\\
1&0&1&0&0&1&1&0&1&0&0&1&0&1\\
1&0&1&0&0&1&0&1&0&1&1&0&1&0\\
1&0&0&1&1&0&1&0&0&1&1&0&0&1\\
1&0&0&1&1&0&0&1&1&0&0&1&1&0\\
1&0&0&1&0&1&1&0&0&1&0&1&1&0\\
1&0&0&1&0&1&0&1&1&0&1&0&0&1\\
0&1&1&0&1&0&1&0&0&1&0&1&1&0\\
0&1&1&0&1&0&0&1&1&0&1&0&0&1\\
0&1&1&0&0&1&1&0&0&1&1&0&0&1\\
0&1&1&0&0&1&0&1&1&0&0&1&1&0\\
0&1&0&1&1&0&1&0&1&0&0&1&0&1\\
0&1&0&1&1&0&0&1&0&1&1&0&1&0\\
0&1&0&1&0&1&1&0&1&0&1&0&1&0\\
0&1&0&1&0&1&0&1&0&1&0&1&0&1
\end{array}\right] \qquad (2.8)$$

$$\underline{R} = \left[\begin{array}{cc|c} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array}\right] \qquad \underline{W} = \left[\begin{array}{cccc|cc} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{array}\right] \qquad (q,n) = (2,2)$$

$$\underline{R} = \left[\begin{array}{cc|c} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \\ 1 & 1 & 1 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \\ 2 & 2 & 2 \end{array}\right] \qquad \underline{W} = \left[\begin{array}{cccccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{array}\right] \qquad (q,n) = (3,2)$$

$$\underline{R} = \left[\begin{array}{cc|c} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & t & t \\ 0 & t^2 & t^2 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & t & t^2 \\ 1 & t^2 & t \\ t & 0 & t \\ t & 1 & t^2 \\ t & t & 0 \\ t & t^2 & 1 \\ t^2 & 0 & t^2 \\ t^2 & 1 & t \\ t^2 & t & 1 \\ t^2 & t^2 & 0 \end{array}\right] \qquad \underline{W} = \left[\begin{array}{cccccccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}\right] \qquad (q,n) = (4,2)$$

$$\underline{R} = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{array}\right] \qquad \underline{W} = \left[\begin{array}{cccccc|cc} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right] \qquad (q,n) = (2,3)$$

RB-3196-64

FIG. 2.6 CODE AND WINDING MATRICES FOR SOME DISTANCE TWO READ ACCESS SWITCHES

The redundant columns of this code are all grouped to the right of the partition in order to show the relationship of this switch to the switches which were reviewed in Subsection A.

A block code may be characterized by several parameters: the radix q, the total number of bits in a code word (or block) t, the effective number of nonredundant bits in each block n, and the minimum distance d. For the example given above $q = 2$, $t = 7$, $n = 4$, $d = 3$. On Fig. 2.7 the parameters t versus n are plotted for the well-known Hamming codes ($d = 3$, $q = 2$).

It is possible to design an access switch, either of the input logic or output logic type, which corresponds to a block code, by forming the $\underline{R}$ matrix of all code words. Then the $\underline{W}$ matrix is written out following the rules given before. The bias winding for this switch must have a sufficient number of turns to reduce the most positive nonselect magnetomotive force to zero. But as there are t ONES in each row of $\underline{W}$ and the code is of distance d, this bias winding must be

$$b = d - t. \tag{2.9}$$

The total number of windings on each core for an input logic type switch is

$$e_i = |b| + t = 2t - d \text{ for } t \geq 3 \tag{2.10}$$

For the output logic type switch, the number of windings on each core is

$$e_o = q^{n-1} \tag{2.11}$$

Values of b, e_o and e_i are given for $q = 2$, $d = 3$ switches on Fig. 2.7. It is interesting to compare e_i and e_o for these read access switches.

Both in this and the previous subsection the method for producing a reliable read access switch has been first to write the $\underline{R}$ matrix which consists of all code words, and then by the defined relationship between the $\underline{R}$ and $\underline{W}$ matrices to write the $\underline{W}$ matrix for the desired switch. Therefore, for a distance d radix q code consisting of N words, there cannot be found in the $\underline{R}$ matrix two code words which differ in fewer than d of the t digits. Then let two rows be chosen which differ in exactly d digits and consider the two rows of $\underline{W}$ which correspond to such a pair of rows in $\underline{R}$: let these be described by the row vectors $\underline{w}_i$ and $\underline{w}_j$. Then by the relationship between $\underline{R}$ and $\underline{W}$, the scalar product of either row with itself is

$$\underline{w}_i \underline{w}_i^T = \underline{w}_j \underline{w}_j^T = t$$

and the scalar product of both rows of $\underline{W}$ is

$$\underline{w}_i \underline{w}_j^T = \underline{w}_j \underline{w}_i^T = t - d.$$

Furthermore, let the read access switch bias be as given in Eq. (2.9). Then each element on the major diagonal of the read access switch excitation matrix $\underline{X}$ is the scalar product of each row of $\underline{W}$ with itself plus the bias, or

$$x_{ii} = x_{jj} = t + d - t = d.$$

But by read access switch construction, every row of $\underline{W}$ has the same number of ONES. Therefore, all elements on the major diagonal have the same value; in this case

$$x_{kk} = d, \text{ for } k = 1, 2, \ldots, N.$$

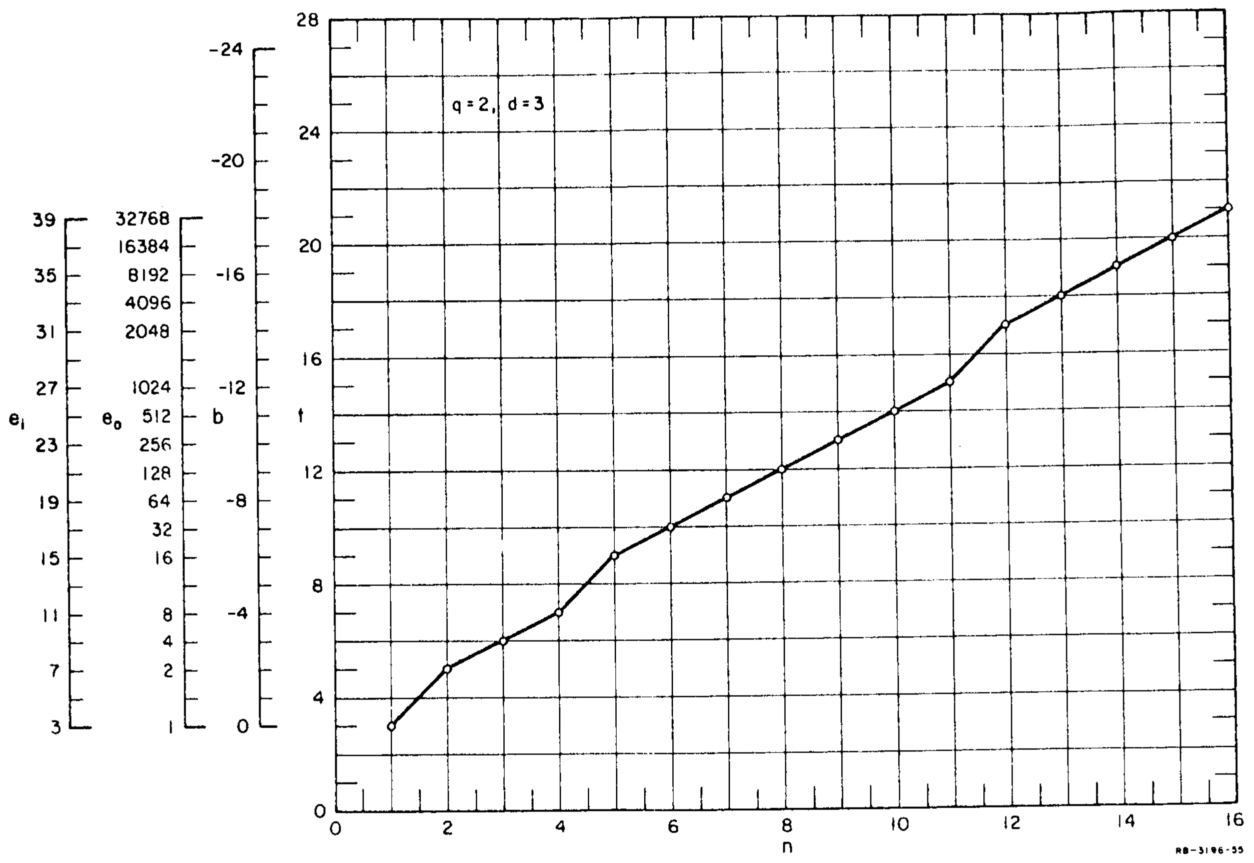


FIG. 2.7 PROPERTIES OF THE HAMMING CODES

The two elements of $\underline{X}$, which correspond to the two rows of $\underline{W}$ which are being considered, are the scalar products of the two rows plus the bias:

$$x_{ij} = x_{ji} = t - d + d - t = 0.$$

But these two rows of $\underline{W}$ correspond to two rows of $\underline{R}$ which differ in the fewest possible number of digits, namely d. Therefore, $\underline{w}_i$ and $\underline{w}_j$ differ in the fewest possible number of digits, and x_{ij} and x_{ji} correspond to the most positive off-diagonal elements of $\underline{X}$.

Therefore, by the definition given in Subsection A, the load-sharing factor of this switch is d. Furthermore, a fundamental relation has been shown:

Given any block code of any radix and having a Hamming distance d, a read access switch may be constructed which has a load-sharing factor d.

In more detail, the correspondence between block codes and read access switches is

Parameter	Code	Switch
q	Radix	Number of input wires per input set.
t	Number of symbols in a code word	Number of sets of input wires.
n	Number of nonredundant symbols in a code word	Number of nonredundant sets of input wires (the dimension).
d	Hamming distance	Load-sharing factor

For later comparisons, it is desirable to consider switches of this type for which $q = 2$, $d = 3$. The properties of these switches are found from Eqs. (2.9), (2.10) and (2.11) to be

$$d = 3, \; b = 3 - t, \; e_i = 2t - 3, \; e_o = 2^{n-1} \tag{2.12}$$

D. RADIX TWO READ ACCESS SWITCHES

A particularly interesting specialization of the switches discussed in the previous subsection is that for $q = 2$: the *radix two* switch. As there are two drivers in each of the t sets of drivers for a radix two switch, each bit of the address code serves to energize one or the other of a pair of drivers. Therefore, if the address code comes from a flip-flop register, no recoding equipment is necessary for a radix two switch.

For radix two switches, the types of failure may be considered in more detail. Up to this point it has been assumed the read access switch failures can occur only by semiconductor drivers open-circuiting. Suppose that in addition to f_1 such open-circuit failures, there are f_0 failures due to driver short-circuits. By the use of appropriate current limiting, it is assumed that a short-circuited driver delivers a known current; let this be the same current as is delivered by a properly-operating energized driver.

The address code which appears at the input of the read access switch may have one or more errors by the complement of the correct bit being present in certain positions. In some cases it may be desirable to correct any such errors before applying the code to the switch; however, it may be advantageous to mask such errors by an appropriate design of the switch. Let the number of address code errors by f_a.

Then for address code failures, the selecting magnetomotive force is reduced in the worst case by f_a, while the worst case nonselecting magnetomotive force is increased by f_a. Then if the bias for such a switch is chosen to be smaller than the previous design value by an amount $-f_a$, the switch operates properly with a new selecting magnetomotive force of $d-2f_a$, providing this quantity equals or exceeds unity. For driver-open failures no change in bias is needed, but the selecting magnetomotive force is further reduced in the worst case by f_1. For driver-short failures, the selecting magnetomotive force is unchanged, but the nonselecting magnetomotive force is increased in the worst case by f_0. Thus the bias must be chosen to be smaller than the previous design value by a further amount $-f_0$, and the select magnetomotive force is further reduced by f_0.

If these three kinds of errors can occur independently, the bias must be changed by an amount

$$\Delta b = -(f_a + f_0), \tag{2.13}$$

and the switch operates properly providing the worst case selecting magnetomotive force equals or exceeds unity:

$$d - (2f_a + f_0 + f_1) \geq 1. \tag{2.14}$$

From Eqs. (2.9) and (2.13), the bias becomes

$$b = d - t - f_a - f_0. \tag{2.15}$$

To illustrate these considerations, suppose a radix two switch is to be designed which operates properly with one address bit error, one driver open-circuit failure, and one driver short-circuit failure. All three of these failures may occur simultaneously. Then from Eq. (2.14), $d \geq 5$ is necessary.

As a second example, suppose a $d = 3$ switch is being used. Then Eqs. (2.14) and (2.15) show that by making the bias one unit more negative than the design value, the switch will operate properly with one address bit error, or with one driver open-circuit failure, or with one driver short-circuit failure, or with both a driver open-circuit failure and a driver short-circuit failure.

E. TWO-DIMENSIONAL READ ACCESS SWITCHES

As one would expect binary codes to be used in practice, the results of the previous subsection suggest the use of n-dimensional radix 2 switches. However, whether the switch is of the input logic or output logic type, the total number of windings on each core (Eqs. (2.10) and (2.11)) increases with the number of outputs to the point that radix two switches become impractical. For $q = 2$ and $d = 3$ Fig. 2.7 shows that a 64-output output logic switch requires 34 windings on each core, and that a 4096-output input logic switch requires 31 windings per core.

If such a large number of windings cannot be tolerated, or if a very large switch is desired, then other means must be employed. The approach for reducing windings which will be taken in this subsection is to consider switches which are limited to two dimensions. Suppose first that q is prime. Then consider the generator[11] matrix $\underline{G}$:

$$\underline{G} = \left[\begin{array}{cc|cccccc} 1 & 0 & q-1 & q-2 & \cdots & 2 & 1 \\ 0 & 1 & 1 & 1 & & 1 & 1 \end{array}\right] . \tag{2.16}$$

A parity-check[10] matrix, $\underline{H}$ which corresponds to the code defined in Eq. (2.16) is

$$\underline{H} = \left[\begin{array}{cc|cccc} 1 & q-1 & 1 & & & \bigcirc \\ 2 & q-1 & & 1 & & \\ & & & & \ddots & \\ q-1 & q-1 & \bigcirc & & & 1 \end{array}\right] . \tag{2.17}$$

By the definition of $\underline{G}$, the $\underline{R}$ matrix, which is recalled to consist of all code words, is formed by taking all linear combinations, modulo q, of the rows of Eq. (2.16):

$$\underline{R} = \begin{array}{c} \\ q \\ \end{array} \left[\begin{array}{cc|ccccccc} \multicolumn{2}{c|}{\leftarrow 2 \rightarrow} & \multicolumn{7}{c}{\leftarrow q \rightarrow} \\ 0 & 0 & 0 & 0 & - & - & - & 0 & 0 \\ 0 & 1 & 1 & 1 & - & - & - & 1 & 1 \\ 0 & 2 & 2 & 2 & - & - & - & 2 & 2 \\ \multicolumn{2}{c|}{---} & & & - & - & - & & \\ 0 & q-1 & q-1 & q-1 & - & - & - & q-1 & q-1 \\ \hline 1 & 0 & q-1 & q-2 & - & - & - & 2 & 1 \\ 1 & 1 & 0 & q-1 & - & - & - & 3 & 2 \\ 1 & 2 & 1 & 0 & - & - & - & 4 & 3 \\ \multicolumn{2}{c|}{---} & & & - & - & - & & \\ 1 & q-1 & q-2 & q-3 & - & - & - & 1 & 0 \\ \hline 2 & 0 & q-2 & q-4 & - & - & - & 4 & 2 \\ 2 & 1 & q-1 & q-3 & - & - & - & 5 & 3 \\ 2 & 2 & 0 & q-2 & - & - & - & 6 & 4 \\ \multicolumn{2}{c|}{---} & & & - & - & - & & \\ 2 & q-1 & q-3 & q-5 & - & - & - & 3 & 1 \\ \hline \multicolumn{2}{c|}{---} & & & - & - & - & & \\ \hline q-1 & 0 & 1 & 2 & - & - & - & q-2 & q-1 \\ q-1 & 1 & 2 & 3 & - & - & - & q-1 & 0 \\ q-1 & 2 & 3 & 4 & - & - & - & 0 & 1 \\ \multicolumn{2}{c|}{---} & & & - & - & - & & \\ q-1 & q-1 & 0 & 1 & - & - & - & q-3 & q-2 \end{array}\right] . \qquad (2.18)$$

Let two vectors be defined as

$$q \updownarrow \begin{bmatrix} i \\ i \\ --- \\ i \end{bmatrix} = \underline{g}'_i \; , \qquad q \updownarrow \begin{bmatrix} q-i \\ q-i+1 \\ q-i+2 \\ --- \\ q-i-1 \end{bmatrix} = \underline{h}'_i \; . \qquad (2.19)$$

Using these vectors, the $\underline{R}$ matrix of Eq. (2.18) may be rewritten as

$$\underline{R} = \left[\begin{array}{cc|cccccc} \underline{g}_0' & \underline{h}_0' & \underline{h}_0' & \underline{h}_0' & - & - & - & \underline{h}_0' & \underline{h}_0' \\ \underline{g}_1' & \underline{h}_0' & \underline{h}_1' & \underline{h}_2' & - & - & - & \underline{h}_{q-2}' & \underline{h}_{q-1}' \\ \underline{g}_2' & \underline{h}_0' & \underline{h}_2' & \underline{h}_4' & - & - & - & \underline{h}_{q-4}' & \underline{h}_{q-2}' \\ --- & & & & - & - & - & & \\ \underline{g}_{q-1}' & \underline{h}_0' & \underline{h}_{q-1}' & \underline{h}_{q-2}' & - & - & - & \underline{h}_2' & \underline{h}_1' \end{array}\right] \qquad (2.20)$$

Furthermore, the matrices $\underline{g}_i$ and $\underline{h}_i$ may be defined from the vectors $\underline{g}_i'$ and $\underline{h}_i'$, respectively, in the same way that the $\underline{W}$ matrix is defined from the $\underline{R}$ matrix:

$$\begin{array}{c|c} & \xleftrightarrow{\qquad q \qquad} \quad \text{Col } i \\ \updownarrow q & \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ & - & - & - & \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} = \underline{g}_i \quad , \qquad \begin{array}{c|c} & \xleftrightarrow{\qquad q \qquad} \\ & \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ & \cdot & \cdot & \cdot & \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} \\ \\ \\ \\ \end{array} \!\!\!\!\!\!\!\!\!\! \begin{array}{c} \text{Row } i \\ \updownarrow q \end{array} = \underline{h}_i \qquad (2.21)$$

With these definitions, the read access switch winding matrix $\underline{W}$, which corresponds to this code is

$$\underline{W} = \left[\begin{array}{cc|cccccc} \underline{g}_0 & \underline{h}_0 & \underline{h}_0 & \underline{h}_0 & - & - & - & \underline{h}_0 & \underline{h}_0 \\ \underline{g}_1 & \underline{h}_0 & \underline{h}_1 & \underline{h}_2 & - & - & - & \underline{h}_{q-2} & \underline{h}_{q-1} \\ \underline{g}_2 & \underline{h}_0 & \underline{h}_2 & \underline{h}_4 & - & - & - & \underline{h}_{q-4} & \underline{h}_{q-2} \\ - - - & & & & - & - & - & & \\ \underline{g}_{q-1} & \underline{h}_0 & \underline{h}_{q-1} & \underline{h}_{q-2} & - & - & - & \underline{h}_2 & \underline{h}_1 \end{array}\right] \qquad (2.22)$$

The code and switch defined above have the following properties:

$$\begin{array}{lll} t = q + 1, & n = 2, & d = q, \\ b = -1, & e_i = q + 2, & e_o = q. \end{array} \qquad (2.23)$$

However, it is not necessary to use all columns in the code described by Eq. (2.16). Suppose only t total columns are used by deleting any $q + 1 - t$ columns of Eqs. (2.16) and (2.22), together with the corresponding rows of Eq. (2.17). Then the code has the property

$$\begin{array}{lll} t \leq q + 1, & n = 2, & d = t - 1, \\ b = -1, & e_i = t + 1, & e_o = q. \end{array} \qquad (2.24)$$

In order to compare this type of switch with those of the previous subsections, consider the important case of $d = 3$. Eq. (2.24) becomes

$$\begin{array}{lll} t = 4, & n = 2, & d = 3, \\ b = -1, & e_i = 5, & e_o = q. \end{array} \tag{2.25}$$

An interesting comparison results between the distance three, n-dimensional radix 2 switches (Eq (2.12)) and the 2-dimensional, radix q switches (Eq. (2.25)). To compare these two switch types, let the number of outputs be the same; *i.e.*,

$$q^2 = 2^n$$

Then the comparison is given in Table 2.1:

Outputs	n-dim., radix 2				2-dim., radix q			
N	t	b	e_i	e_o	q	b	e_i	e_o
16	7	–4	11	8	4	–1	5	4
64	10	–7	17	32	8	–1	5	8
256	12	–9	21	128	16	–1	5	16
1024	14	–11	25	512	32	–1	5	32
4096	17	–14	31	2048	64	–1	5	64

Table 2.1

From this comparison it appears that for almost all switches an input logic two dimensional switch of radix $q > 2$ is most advantageous. However, if the input code is binary, a recoding network is necessary in order to convert the code radix. As the input code is redundant, this may be done with various types of redundant networks. As such networks have been considered in detail elsewhere[6], they will not be treated here.

F. EXPANDED TWO-DIMENSIONAL READ ACCESS SWITCHES

The n-dimensional radix two read access switches of Subsections C and D required no recoding equipment; however, it was shown that for a large number of outputs they may become impractical. For this reason the two-dimensional radix q switches were developed in Subsection E. These switches were shown to have a constant number of windings per core; however, recoding circuitry is necessary for these switches.

Another approach to the two-dimensional radix q read access switch is to expand the radix q code into binary and to use this code external to the memory. At the expense of a less efficient code, the recoding equipment is made considerably simpler. To illustrate this, consider the case of $N = 16$. From Table 2.1, $q = 4$; therefore, the $\underline{R}$ matrix, $\underline{R}_4$, for this switch is

$$\underline{R}_4 = \left[\begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 2 & 2 & 2 \\ 0 & 3 & 3 & 3 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 3 \\ 1 & 2 & 3 & 0 \\ 1 & 3 & 2 & 1 \\ 2 & 0 & 2 & 3 \\ 2 & 1 & 3 & 2 \\ 2 & 2 & 0 & 1 \\ 2 & 3 & 1 & 0 \\ 3 & 0 & 3 & 1 \\ 3 & 1 & 2 & 0 \\ 3 & 2 & 1 & 3 \\ 3 & 3 & 0 & 2 \end{array}\right] \qquad (2.26)$$

The corresponding winding matrix is

$$\underline{W} = \left[\begin{array}{cccc|cccc|cccc|cccc}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right] . \tag{2.27}$$

External to the memory, the following code is used:

$$\underline{R}_2 = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 0
\end{bmatrix} . \tag{2.28}$$

The code of Eq. (2.28) is produced by replacing each radix 4 column of Eq. (2.26) with two radix 2 columns. The only recoding equipment needed in this case is a single two-input AND gate associated with each driver. The complete address register, recoder, switch combination for this example is drawn as Fig. 2.8.

For the recoding simplification in this example the code requires one additional bit over the best $d = 3$, $q = 2$ block code (see Fig. 2.7).

To generalize this type code, suppose $N = 2^{2k}$ and $q = 2^k$. Substituting this in Eq. (2.24) leads to the properties of this code, where t is the number of radix 2^k columns in the code. Letting t' be the number of binary columns in the expanded code, the result is

$$\begin{array}{llll} t \leq 2^k + 1, & t' = tk, & N = 2^{2k}, & d = t - 1, \\ b = -1, & e_i = t + 1, & e_o = 2^k. & \end{array} \qquad (2.29)$$

For the important case of $d = 3$, Eq. (2.29) reduces to

$$\begin{array}{llll} t = 4, & t' = 4k, & N = 2^{2k}, & d = 3, \\ b = -1 & e_i = 5, & e_o = 2^k. & \end{array} \qquad (2.30)$$

The number of diodes, D, necessary to perform the recoding function for an expanded two-dimensional switch may be determined by noting that there are t radix 2^k columns in the code. Therefore there are $t\ 2^k$ drivers in the switch, exclusive of the bias driver. Each driver is associated with a k-input AND gate; therefore,

$$D = t\, k 2^k.$$

But from Eq. (2.29), $k = 2n$, so

$$D = t\, n\, 2^{\frac{n}{2} - 1}. \qquad (2.31)$$

G. INTERLEAVED READ ACCESS SWITCHES

Another family of switches which may be attractive are those which will be called the *r-interleaved read access switches*. They will be shown to have properties intermediate between the expanded two dimensional switches and the radix two switches. A 2-interleaved code (t', n') is formed from a (t, n) radix 2 block code by writing an $\underline{R}'_2$ matrix which contains in the rows all different pairs of code words from the $\underline{R}$ matrix. These code words are then interleaved by column. Next, a new $\underline{R}'_4$ matrix is written in which pairs of columns are recoded in radix 4. Finally, the $\underline{R}'_4$ matrix is translated into the switch winding matrix by the definitions given before.

An r-interleaved code is similarly formed: the (t, n) radix 2 block code is iterated r times. The $\underline{R}'_2$ matrix is partitioned into groups of r columns, and the $\underline{R}'_{2^r}$ matrix is written in radix 2^r.

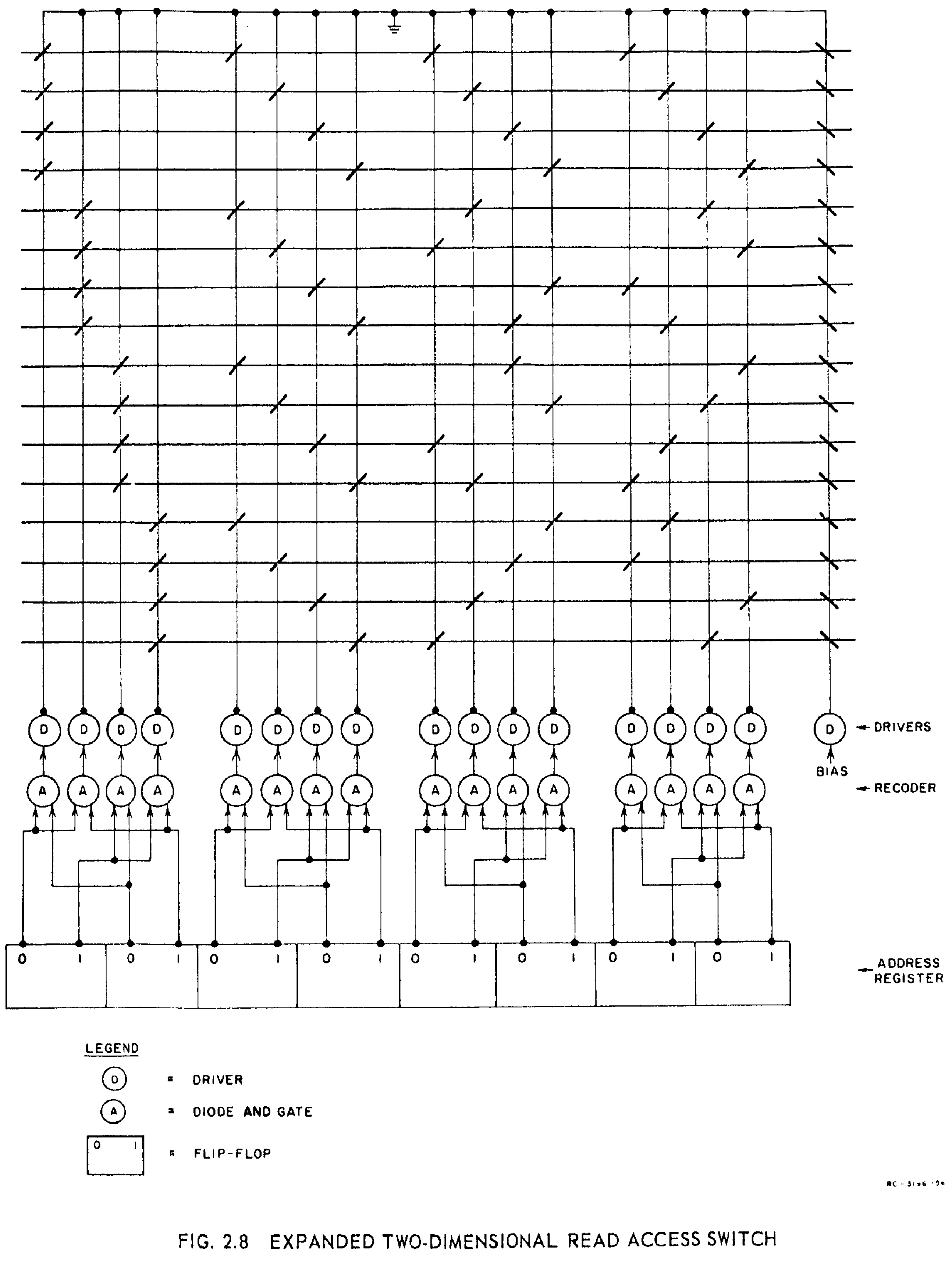


FIG. 2.8 EXPANDED TWO-DIMENSIONAL READ ACCESS SWITCH

An example should clarify the 2-interleaved code: Suppose one starts with a $(t, n) = (5, 2)$ radix two code:

$$\underline{R}(5, 2) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{2.32}$$

The 2-interleaved code has $(t', n') = (10, 4)$; the $\underline{R}'_2$ matrix is

$$\underline{R}'_2\,(10, 4) = \left[\begin{array}{cc|cc|cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \end{array}\right] \tag{2.33}$$

It is partitioned into pairs of columns which then are recoded into radix four as

$$\underline{R}'_4(10,4) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 0 & 2 & 2 \\ 3 & 0 & 0 & 3 & 3 \\ 3 & 1 & 1 & 3 & 2 \\ 2 & 1 & 1 & 2 & 3 \\ 2 & 2 & 2 & 2 & 0 \\ 3 & 2 & 2 & 3 & 1 \\ 3 & 3 & 3 & 3 & 0 \\ 2 & 3 & 3 & 2 & 1 \\ 0 & 2 & 2 & 0 & 2 \\ 1 & 2 & 2 & 1 & 3 \\ 1 & 3 & 3 & 1 & 2 \\ 0 & 3 & 3 & 0 & 3 \end{bmatrix} \tag{2.34}$$

The corresponding winding matrix is

$$\underline{W}' = \left[\begin{array}{cccc|cccc|cccc|cccc|cccc}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right] \tag{2.35}$$

By inspection, the distance of this (10, 4) 2-interleaved code is $d = 3$; this is the same as the constituent (5, 2) codes. This property can be shown to be valid in general for r-interleaved codes.

It is easily seen that for any r-interleaved code, based on r iterations of a (t, n) code, there are t ONES in each row of $\underline{W}'$. Therefore, the arguments which led to Eq. (2.9) are valid here. Similarly, Eqs. (2.10) and (2.11) hold. Therefore, letting the r-interleaved code (t', n') be based on a (t, n) code of distance d, the properties are found to be

$$\begin{array}{llll} t' = r\,t, & n' = r\,n, & d' = d, & \\ b' = d' - \dfrac{t'}{r}, & e'_i = \dfrac{2t'}{r} - d', \ (\text{for } t' \geq r\,d'), & e'_o = 2^{n' - r}. & \end{array} \tag{2.36}$$

For the special case $d = 3$, $r = 2$,

$$\begin{array}{lll} t' = 2t, & n' = 2n, & d' = 3, \\ b' = 3 - \dfrac{t'}{2}, & e'_i = t' - 3, & e'_o = 2^{n' - 2}, \end{array} \tag{2.37}$$

and for the special case $d = 3$, $r = 3$,

$$\begin{array}{lll} t' = 3t, & n' = 3n, & d' = 3, \\ b' = 3 - \dfrac{t'}{3}, & e'_i = \dfrac{2t'}{3} - 3, & e'_o = 2^{n' - 3}. \end{array} \tag{2.38}$$

The number of recoder diodes required for a switch to be used with an r-interleaved code may be determined by noting that the $\underline{R}'_{2^r}$ matrix is written in radix 2^r and has t columns. Therefore, there are $t2^r$ switch drivers (exclusive of the bias). Each driver has an r-input AND gate associated with it; consequently the number of recoder diodes is

$$D = r\,t2^r \tag{2.39}$$

H. COMPARISON OF THE RESULTS

In order to give a means for comparison of the various codes and switches, Figs. 2.9 through 2.13 have been drawn. The abscissa for all these figures is n, the number of nonredundant bits that would be necessary for the code. The five parameters t, b, e_i, e_o, and D are plotted for four switches:

Radix two - Described by Eq. (2.12)
2-Interleaved - Described by Eq. (2.37)
3-Interleaved - Described by Eq. (2.38)
Expanded two-dimensional - Described by Eq. (2.30)

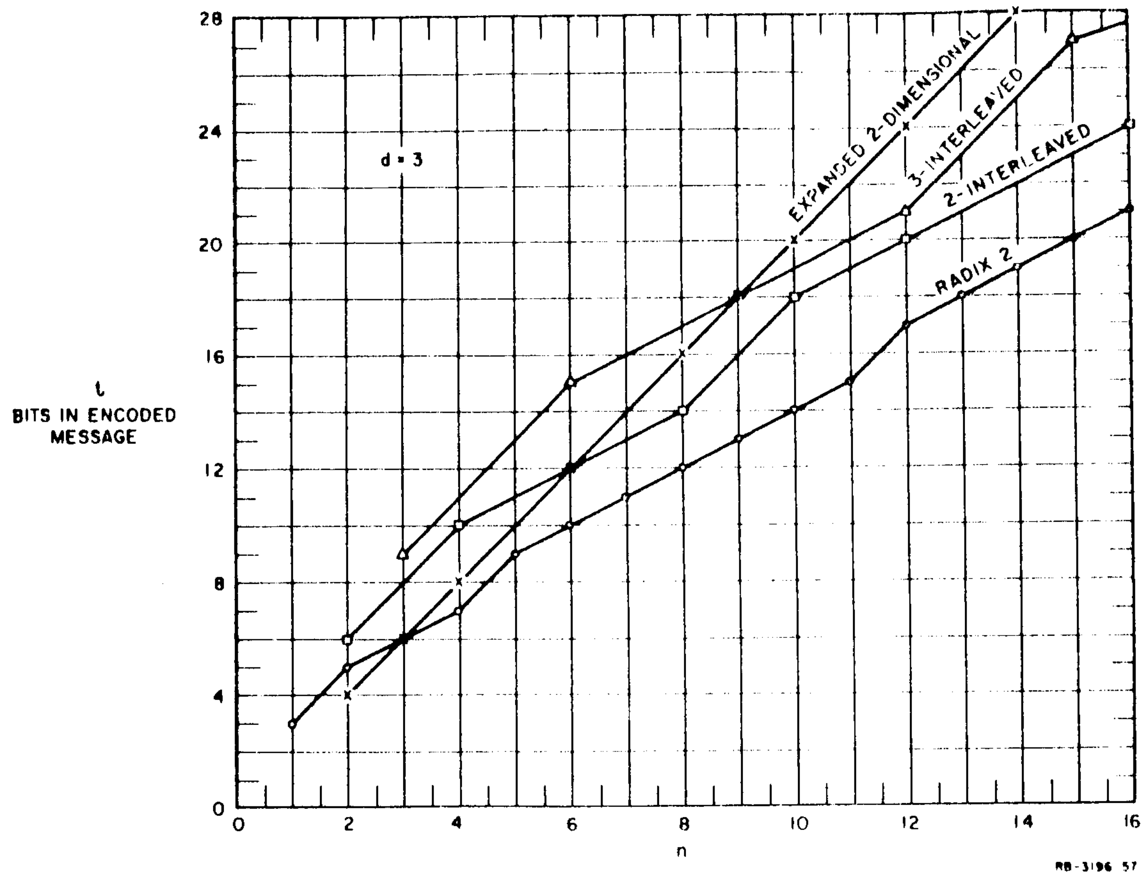


FIG. 2.9 THE NUMBER OF ENCODED MESSAGE BITS vs. THE NUMBER OF NONREDUNDANT BITS FOR VARIOUS d 3 READ ACCESS SWITCHES

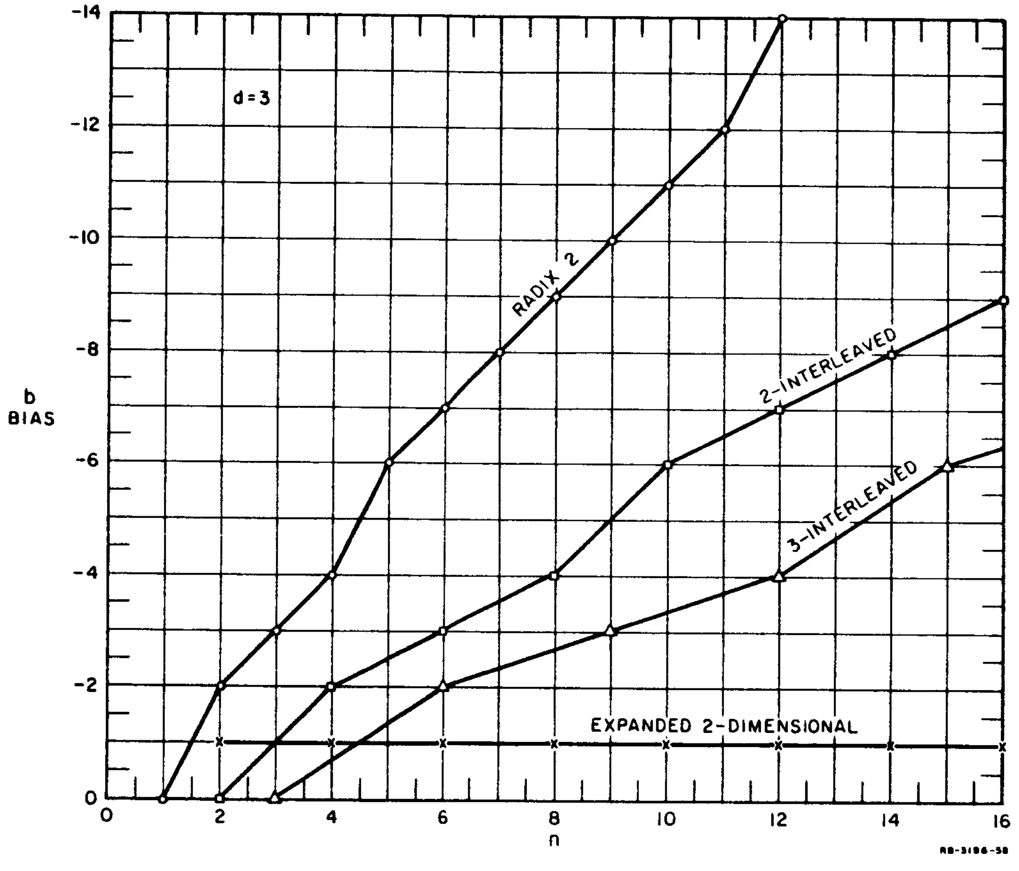


FIG. 2.10 THE BIAS vs. THE NUMBER OF NONREDUNDANT BITS FOR VARIOUS d=3 READ ACCESS SWITCHES

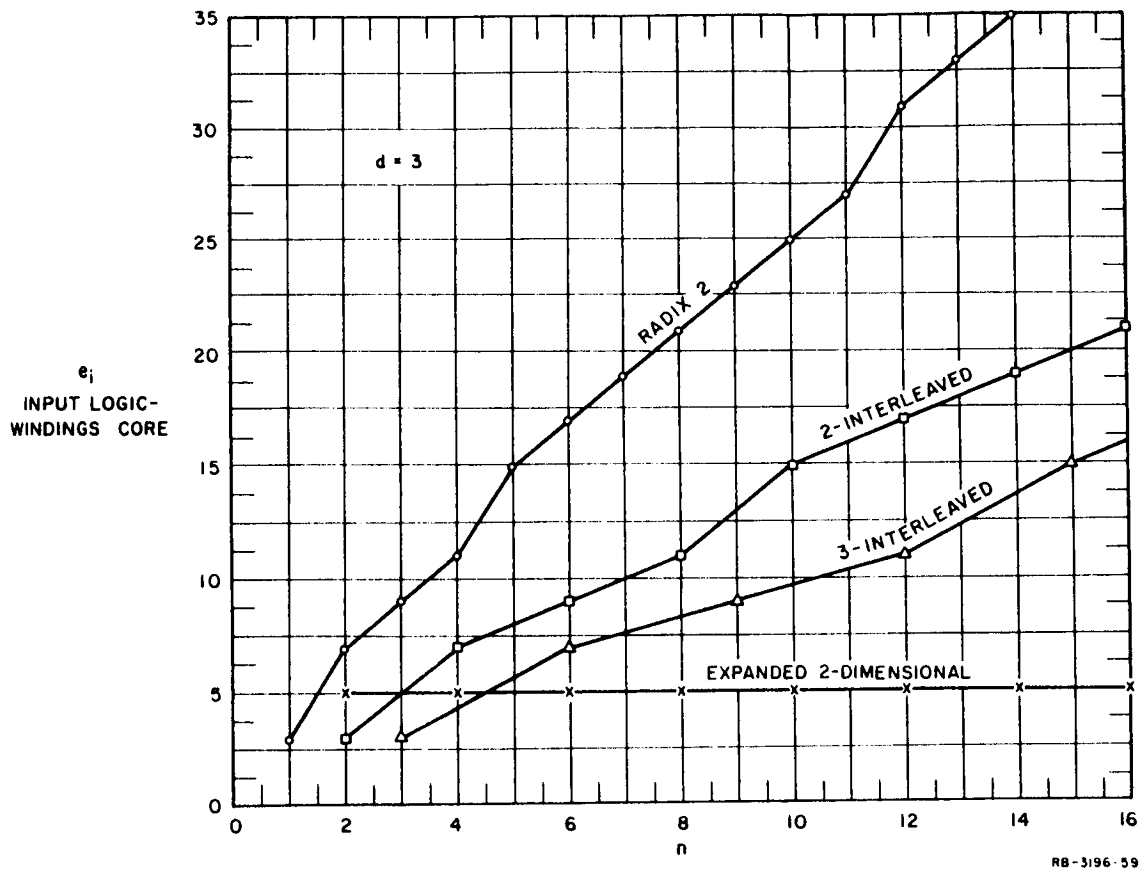


FIG. 2.11 THE NUMBER OF WINDINGS PER CORE vs. THE NUMBER OF NONREDUNDANT BITS FOR VARIOUS d=3 INPUT LOGIC TYPE READ ACCESS SWITCHES

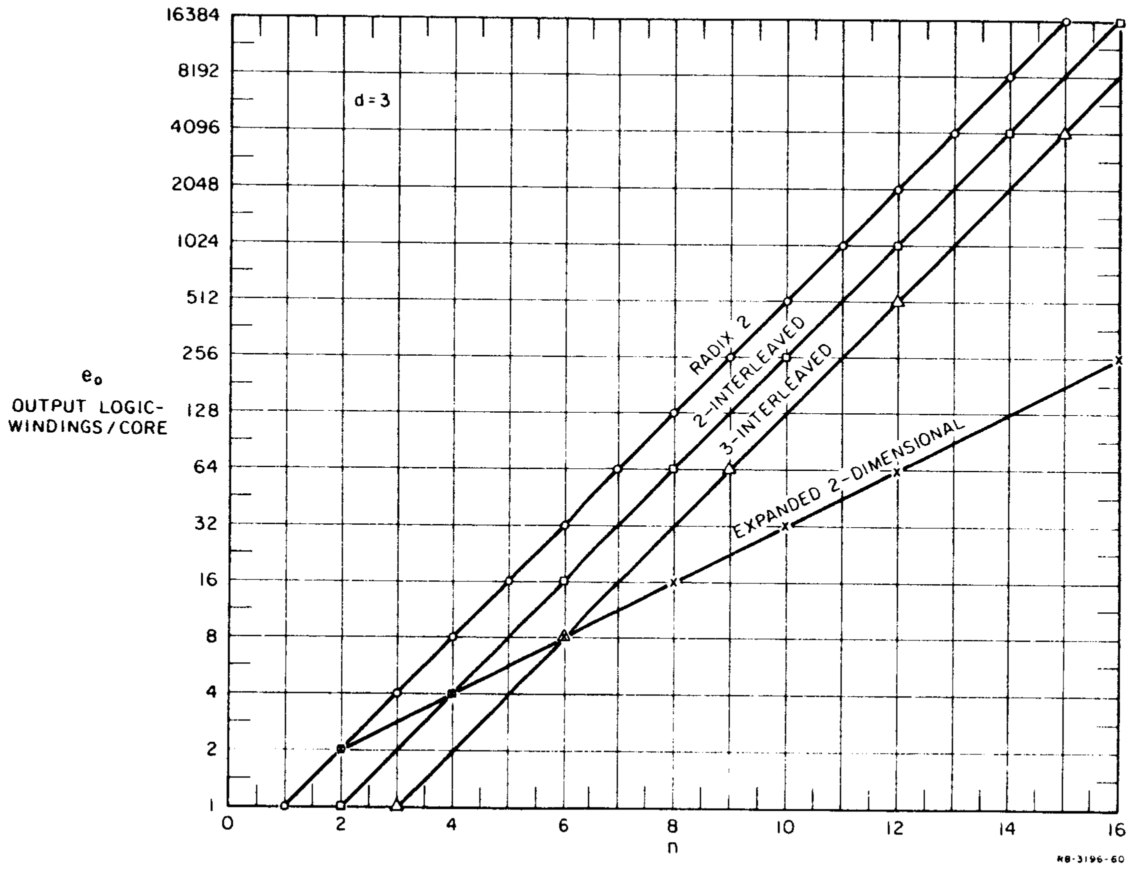


FIG. 2.12 THE NUMBER OF WINDINGS PER CORE vs. THE NUMBER OF NONREDUNDANT BITS FOR VARIOUS d=3 OUTPUT LOGIC TYPE READ ACCESS SWITCHES

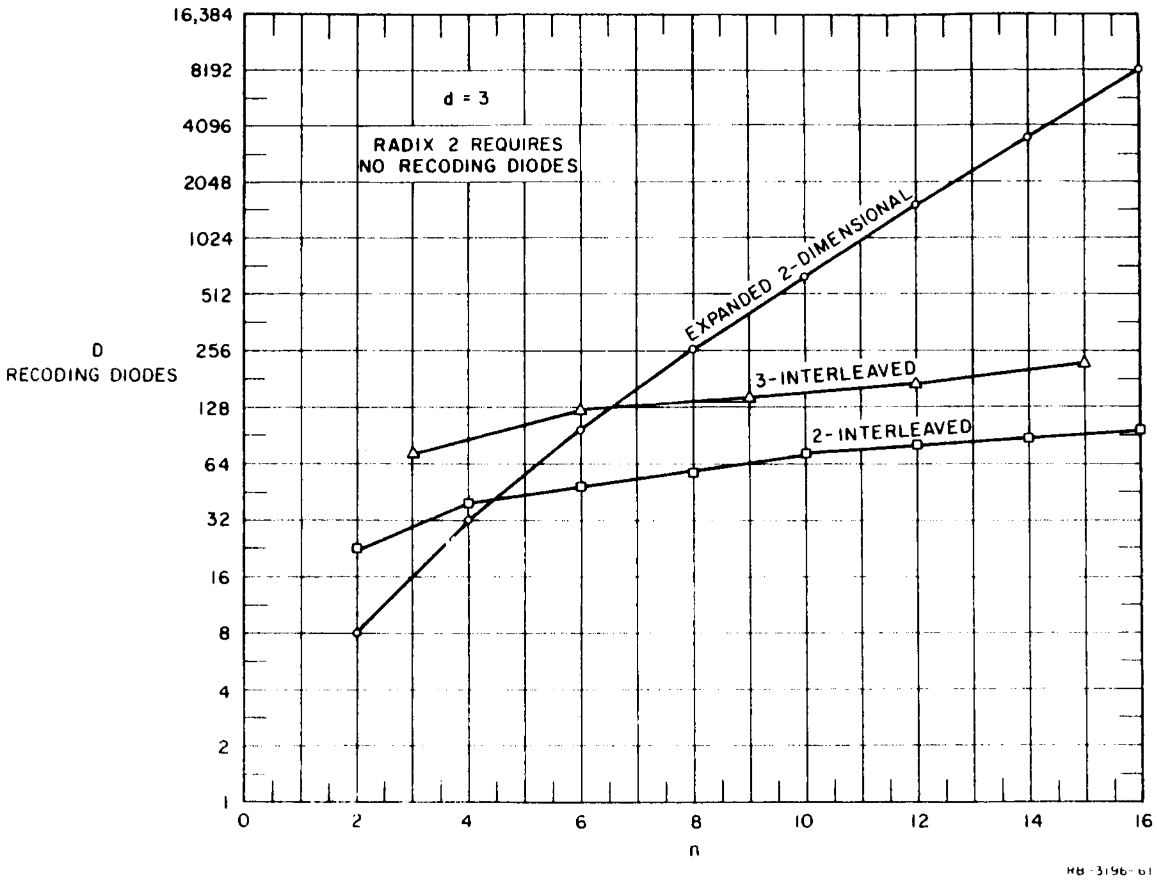


FIG. 2.13 THE NUMBER OF RECODING DIODES vs. THE NUMBER OF NONREDUNDANT BITS FOR VARIOUS d=3 READ ACCESS SWITCHES

All of these switches and codes are $d = 3$, and are based on the Hamming codes. It is seen from these figures that the radix two codes have the lowest values of t for a given value of n; however, they also have the largest number of windings per core. The 2-interleaved codes have a slight increase in the number of message bits. This plus a small amount of recoding equipment is the price paid for a reduction in the number of windings per core. Similarly, a 3-interleaved code further reduces the number of windings per core by increasing the recoding equipment and the bits in the encoded message. And finally, the expanded 2-dimensional code has a constant number of windings per core (for the input logic case) by using still more recoding equipment and twice as many bits in the redundant message as in the nonredundant message.

While they are not drawn on Figs. 2.9 through 2.13, it is clear that r-interleaved codes for $r > 3$ may be included.

The choice of code and read access switch for a given application involves many considerations which cannot be anticipated, such as the costs and reliability of the various components and the requirements on the code external to the memory system. However, it seems evident that given the design requirements a "best" redundant read access switch, or at least a good approximation to it, can be found.

III WRITE ACCESS SWITCHES

It is recalled that the write access switch receives an input code word, and under certain failure conditions delivers proper write currents on the several wires W_i. Therefore, the write access switch is a combination of the normal inhibit drivers for a magnetic core memory and fault-masking circuitry. Continuing the design philosophy that has been assumed, it is not desirable to increase the number of semiconductor drivers unduly; however, a complication of the magnetic circuitry is allowed.

A general theory of write access switches will not be developed for the present report; rather, a simple example of such a switch will be developed to indicate the possibilities. Suppose that the only error that can occur is a single open-circuit failure of a write access switch driver, and that a five-bit code is used. Such a code, which is termed asymmetric,[6] is shown plotted on the five-cube of Fig. 2.14. The vertices indicated with squares indicate correct code words. Those vertices indicated with circles and connected with a square by a heavy line are incorrect code words which might occur upon a single open-circuit failure. The vertices are chosen so that if any circled vertex occurs upon a single open-circuit failure, only one square vertex is unit distance from it, and below; this is accomplished by disallowing those vertices marked with triangles.

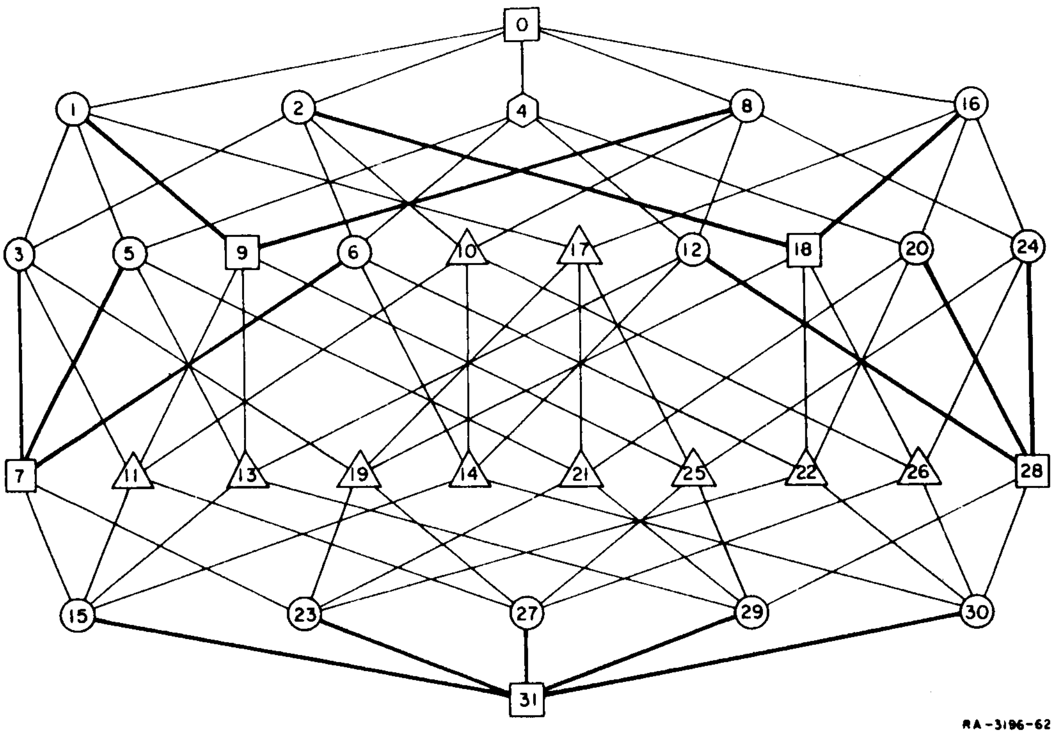


FIG. 2.14 A FIVE-CUBE ILLUSTRATING AN ASYMMETRIC CODE

Therefore, the six valid five-bit code words are

Vertex	y_5	y_4	y_3	y_2	y_1
0	0	0	0	0	0
9	0	1	0	0	1
18	1	0	0	1	0
7	0	0	1	1	1
28	1	1	1	0	0
31	1	1	1	1	1

There is some variation allowed in the choice of the valid code words. For instance, it is seen from Fig. 2.14 that the Vertex 4 could be substituted for the Vertex 0.

Let a write access switch be designed which has as its inputs, x_5, . . ., x_1, and as its outputs, y_5, . . ., y_1. A truth table for this switch is constructed from Fig. 2.14, and is shown as Fig. 2.15. The blank rows correspond to vacuous cases. Taking these vacuous cases into account, the following switching functions result:

x_5	x_4	x_3	x_2	x_1	y_5	y_4	y_3	y_2	y_1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	1
0	0	0	1	0	1	0	0	1	0
0	0	0	1	1	0	0	1	1	1
0	0	1	0	0					
0	0	1	0	1	0	0	1	1	1
0	0	1	1	0	0	0	1	1	1
0	0	1	1	1	0	0	1	1	1
0	1	0	0	0	0	1	0	0	1
0	1	0	0	1	0	1	0	0	1
0	1	0	1	0					
0	1	0	1	1					
0	1	1	0	0	1	1	1	0	0
0	1	1	0	1					
0	1	1	1	0					
0	1	1	1	1	1	1	1	1	1
1	0	0	0	0	1	0	0	1	0
1	0	0	0	1					
1	0	0	1	0	1	0	0	1	0
1	0	0	1	1					
1	0	1	0	0	1	1	1	0	0
1	0	1	0	1					
1	0	1	1	0					
1	0	1	1	1	1	1	1	1	1
1	1	0	0	0	1	1	1	0	0
1	1	0	0	1					
1	1	0	1	0					
1	1	0	1	1	1	1	1	1	1
1	1	1	0	0	1	1	1	0	0
1	1	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

RB-3196-63

FIG. 2.15 TRUTH TABLE FOR A WRITE ACCESS SWITCH

$$\begin{aligned} y_1 &= x_1 + x_2(x_3 + x_4) + x_3'x_4x_5' \\ y_2 &= x_2 + x_1(x_3 + x_5) + x_3'x_4'x_5 \\ y_3 &= x_3 + x_2(x_1 + x_4) + x_4x_5 \\ y_4 &= x_4 + x_5(x_1 + x_3) + x_1x_2'x_3' \\ y_5 &= x_5 + x_4(x_2 + x_3) + x_1'x_2x_3' \end{aligned} \tag{2.40}$$

Now if each of these switching functions could be implemented with a single magnetic core, a simple write access switch would result. This could be done if the functions were linearly-separable[14]; however, it can be shown that for this example they are not. Each of the switching functions of Eq. (2.40) may be divided into two linearly-separable functions in a number of ways. For instance

$$\begin{array}{lll} y_1 = y_{11} + y_{12}, & y_{11} = x_1 + x_2(x_3 + x_4), & y_{12} = x_1'x_3'x_4x_5' \\ y_2 = y_{21} + y_{22}, & y_{21} = x_2 + x_1(x_3 + x_5), & y_{22} = x_2'x_3'x_4'x_5 \\ y_3 = y_{31} + y_{32}, & y_{31} = x_3 + x_2(x_1 + x_4), & y_{32} = x_2'x_3'x_4x_5 \\ y_4 = y_{41} + y_{42}, & y_{41} = x_4 + x_5(x_1 + x_3), & y_{42} = x_1x_2'x_3'x_4' \\ y_5 = y_{51} + y_{52}, & y_{51} = x_5 + x_4(x_2 + x_3), & y_{52} = x_1'x_2x_3'x_5' \end{array} \tag{2.41}$$

By comparing Eqs. (2.40) and (2.41) it appears that the functions y_{i2} are more complicated than necessary. This is done so that the functions y_{i1} and y_{i2} are disjoint at non-vacuous vertices; that is,

$$y_{i1}y_{i2} = 0 \quad \text{for } i = 1, \ldots, 5, \text{ except possibly at vertices } 0, 10, 11, 13, 14, 17, 19, 21, 22, 25, 26 \tag{2.42}$$

A magnetic circuit to implement Eq. (2.41) is shown as Fig. 2.16. Each output wire in this figure threads two cores: the top core of each pair produces y_{i1}, and the bottom core produces y_{i2}. Because of Eq. (2.42), only the top core in each pair is active if all drivers are operating properly; in this case the appropriate outputs are selected with a load-sharing factor of $d = 2$. If any one driver fails by open-circuiting, the network masks this failure by energizing the proper y_i wires. The load-sharing factor is reduced when a driver fails.

It seems evident that the techniques which have been developed for the present example of a write access switch can be extended. While an asymmetric code has been used in order to prevent the example from becoming too large, the extension to other codes should be straightforward.

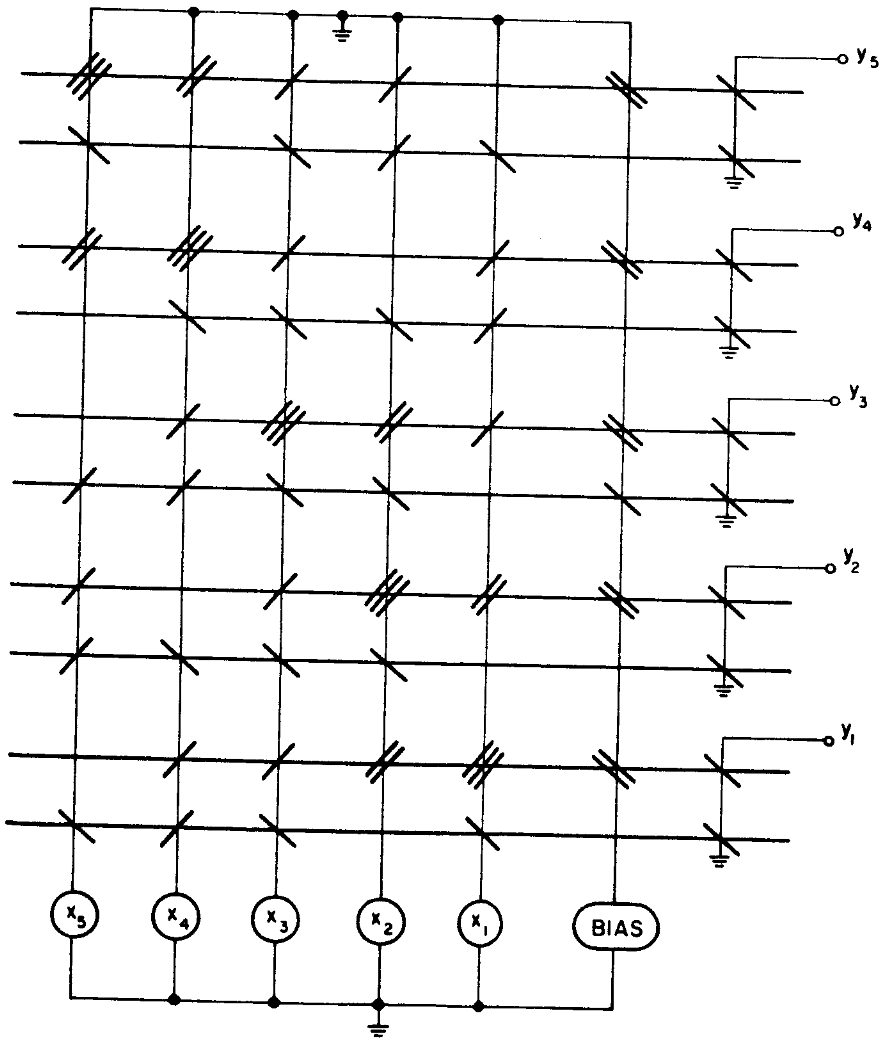


FIG. 2.16 A d=2 WRITE ACCESS SWITCH

IV CURRENT-LIMITING TECHNIQUES

The various magnetic devices which have been considered in Secs. II and III have the property of masking certain kinds of failures in the semiconductor drivers. However, under various failure conditions the delivered power may vary over a considerable range. As was mentioned in Sec. I, if the memory elements are multi-apertured cores the currents delivered by the read and write access switches need not be carefully controlled. However, if single-apertured cores are used, which approach has a number of advantages, these currents must be carefully controlled.

A method by which current variations may be limited is to choose a load-sharing factor which is greater than the minimum necessary to mask the worst failure conditions to be expected. For instance, suppose a read access switch is considered which is designed to tolerate a single driver open-circuit failure. Then Eq. (2.14) indicates that in the failure condition the load-sharing factor is reduced from d to $d - 1$. If $d = 2$, this means a 2:1 change in the load current, while if $d = 5$ (for instance), the load current varies in the ratio 5:4.

As it is impractical to choose very large values of d in order to obtain sufficient current regulation for proper operation, other current limiting methods are indicated. These other methods may be used alone or together with the above method. As current-limiters should not of themselves be subject to failures, semiconductor devices are ruled out. A particularly simple and reliable passive current-limiter is a rectangular hysteresis loop magnetic core acting as a nonlinear load. To see how such a current-limiter would work, consider the circuit drawn in Fig. 2.17. Core A is intended to represent a typical switch core in a read or write access switch. The applied magnetomotive force to this core is

$$H = N_{a1} I_a \tag{2.43}$$

Current I_b is delivered to the memory cores, which are connected in series with a swamping resistance. This combination is indicated by R in Fig. 2.17. Also in series is Core B, which acts as a nonlinear load. Core A has a switching resistance of r_a, and a threshold magnetomotive force H_{oa}; Core b similarly is specified by r_b and H_{ob}. Flux considerations are neglected in the present treatment.

The voltage equations for the two cores are

$$\begin{aligned} E_a &= N_a r_a (H - H_{oa} - N_a I_b) \\ E_b &= N_b r_b (N_b I_b - H_{ob}), \end{aligned} \tag{2.44}$$

but the coupling-loop equation is

$$I_b R = E_a - E_b. \tag{2.45}$$

Combining these equations and simplifying,

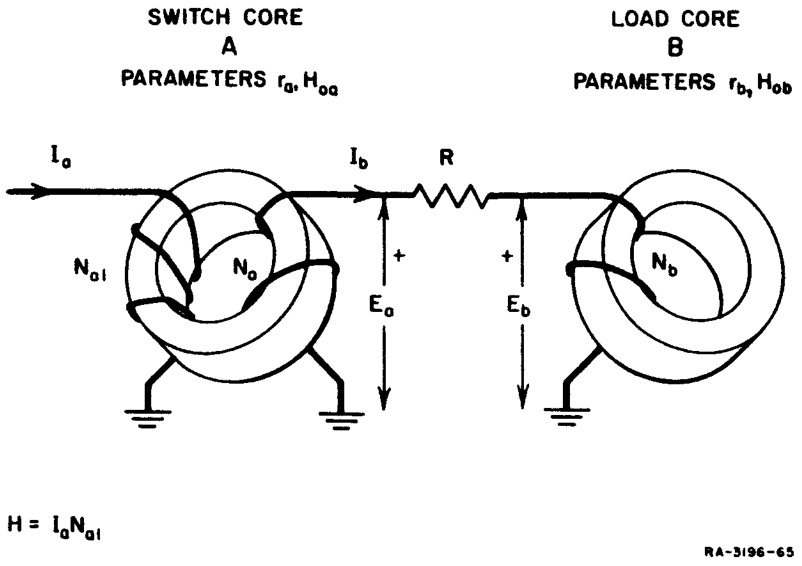


FIG. 2.17 A LOAD CORE CIRCUIT

$$I_b = \frac{N_a r_a (H - H_{oa}) + N_b r_b H_{ob}}{R + N_a^2 r_a + N_b^2 r_b} \quad . \tag{2.46}$$

Eq. (2.46) is valid only if the threshold magnetomotive force of Core B is equalled or exceeded; that is, if

$$N_b I_b \geq H_{ob} \quad . \tag{2.47}$$

Combining Eqs. (2.46) and (2.47), Eq. (2.46) is valid for

$$H \geq H_{oa} + H_{ob} \left(\frac{N_a}{N_b} + \frac{R}{N_a N_b r_a} \right) \equiv H'. \tag{2.48}$$

For values of the applied magnetomotive force less than specified by Eq. (2.48), Core B is effectively out of the circuit, and so I_b is determined by Eq. (2.46) with $N_b = 0$:

$$I_b = \frac{N_a r_a (H - H_{oa})}{R + N_a^2 r_a} \quad . \tag{2.49}$$

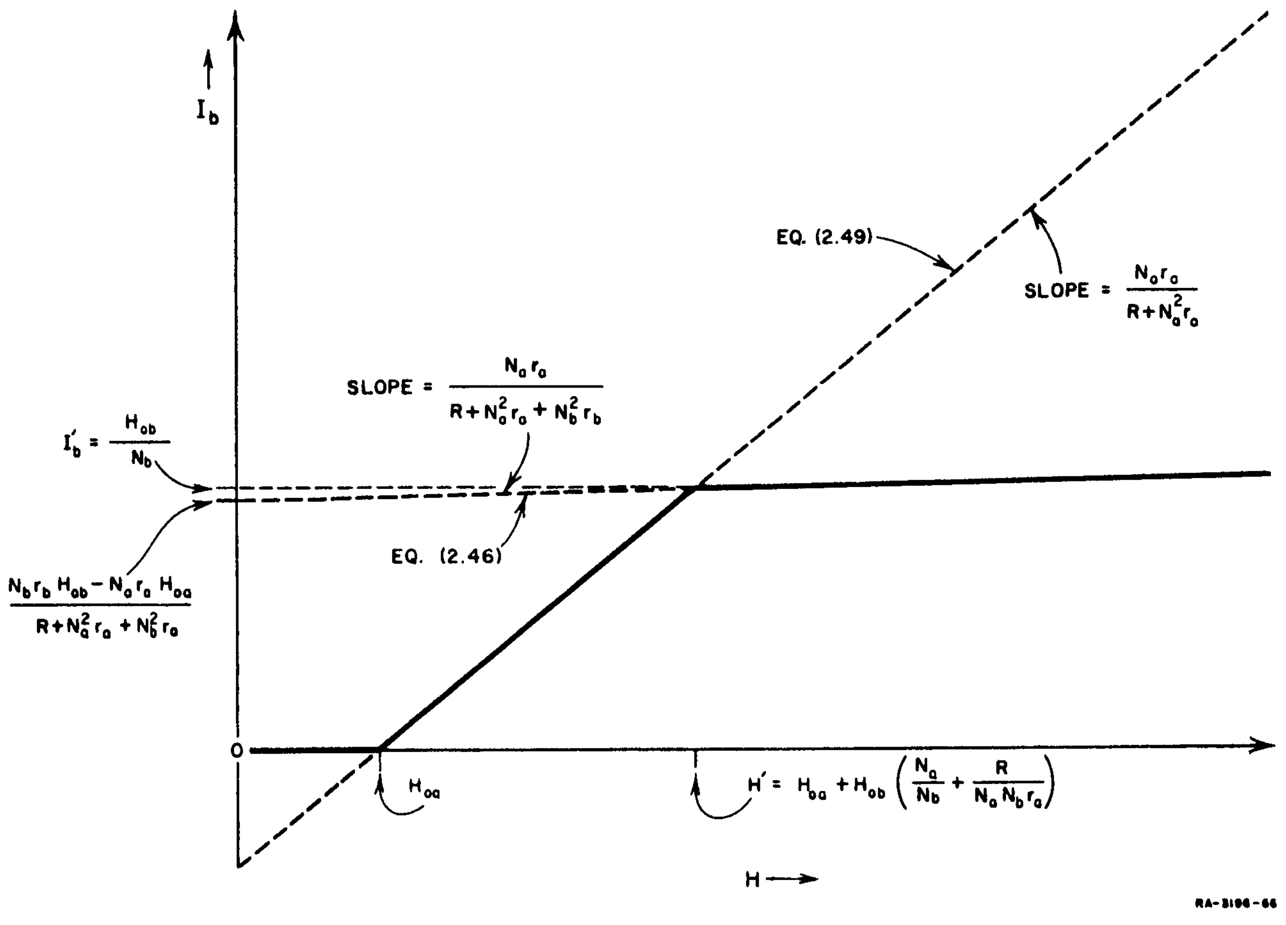


FIG. 2.18 THE SECONDARY CURRENT vs. THE APPLIED MAGNETOMOTIVE FORCE FOR THE CIRCUIT OF FIG. 2.17

But Eq. (2.49) is valid only if the threshold magnetomotive force of Core A is equalled or exceeded; that is, if

$$H \geq H_{oa} \, . \tag{2.50}$$

At values of H less than specified by Eq. (2.50), $I_b = 0$.

Equations (2.46), (2.48), (2.49) and (2.50) are used in Fig. 2.18 to plot the delivered current versus the applied magnetomotive force. The heavy line indicates the delivered current; the dotted lines are included to show the construction. Typically one would design the system so that under a failure condition that reduced the load-sharing factor to one, the applied magnetomotive force in Fig. 2.18 would be H' and the delivered current would be I_b'.

Suppose an access switch nominally operates with a load-sharing factor of d, but under certain failure conditions the load-sharing factor may be reduced to one. Then the fractional variation in output current, k, is

$$k\, I_b\,(H = H') \;=\; I_b(H = H'd) - I_b(H = H') \, .$$

Using Eqs. (2.46) and (2.48) and simplifying,

$$\frac{k}{d-1} = \frac{N_a N_b r_a H_{oa} + (N_a{}^2 r_a + R)H_{ob}}{(N_b{}^2 r_b + N_a{}^2 r_a + R)H_{ob}} \, . \tag{2.51}$$

By using the expected sizes of the various quantities,

$$R \ll N_b{}^2 r_b,\; N_a{}^2 r_a \ll N_b{}^2 r_b,\; N_a r_a H_{oa} \ll N_b r_b H_{ob}.$$

Eq. (2.51) may be approximated by

$$\frac{k}{d-1} \doteq \frac{N_a r_a H_{oa}}{N_b r_b H_{ob}} + \frac{N_a{}^2 r_a}{N_b{}^2 r_b} + \frac{R}{N_b{}^2\, r_b} \, . \tag{2.52}$$

From Fig. 2.18, the nominal current delivered to the cores is

$$I_b' = \frac{H_{ob}}{N_b} \, . \tag{2.53}$$

Therefore, combining Eqs. (2.52) and (2.53), and solving for N_b,

$$N_b = +\sqrt{\left[\frac{d-1}{k\, r_b}\right] \left[\frac{N_a r_a H_{oa}}{I_b'} + N_a{}^2 r_a + R\right]} \tag{2.54}$$

And, from Eq. (2.53)

$$H_{ob} = N_b I_b' \qquad (2.55)$$

Equations (2.54) and (2.55) enable the designer to choose the number of turns and the threshold magnetomotive force of the load core in accordance with the design requirements. To illustrate this, suppose a load core is needed for a switch having $N_a = 10$ turns, $H_{oa} = 0.5$ ampere-turns, and $R = 10$ ohms. Suppose the delivered current should be 200 milliamperes if the load-sharing factor is reduced to one, and that it must not be more than 5 percent higher (*i.e.*, 210 ma.) when the load-sharing factor is the normal value of $d = 2$. Let $r_a = 1$ ohm. Then from Eqs. (2.54) and (2.55)

$$\begin{aligned} N_b &= 52/\sqrt{r_b} \\ H_{ob} &= 10.4/\sqrt{r_b} \end{aligned} \qquad (2.56)$$

A typical ferrite core having a diameter of 1 inch and a cross-sectional area of 0.1 inch has a switching resistance of about 4.7 ohms. With this size, Eq. (2.56) becomes

$$\begin{aligned} N_b &= 24 \text{ turns} \\ H_{ob} &= 4.8 \text{ ampere-turns} \end{aligned} \qquad (2.57)$$

Other core shapes are possible as long as the product of the diameter and the cross-sectional area remains constant.

While a load core is needed for each switch core in a write access switch, it may be possible to have a single load core associated with an entire read access switch. Such a connection is shown in Fig. 2.19. It should be pointed out, however, that the back voltage produced on unselected memory lines by the load core may require the use of a separate load core for each read access switch core.

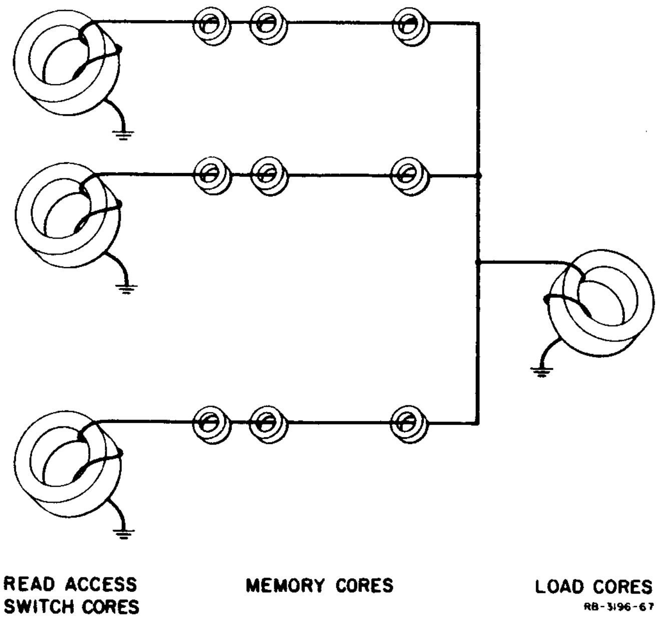


FIG. 2.19 A LOAD CORE CIRCUIT FOR THE READ ACCESS SWITCH

V CONCLUSIONS

A number of techniques have been developed for improving the reliability of a random-access magnetic core memory. The basic philosophy which was adopted in developing these techniques is that semiconductors are more likely to fail than are passive magnetic devices. Consequently the methods which have been considered mask certain faults in the system produced by semiconductor failures, as well as some faults in the input data and produced externally to the memory system.

Read access switches have been considered in some detail. In Sec. II a number of alternative choices are given the designer for picking a read access switch to match the incoming code. Or, if the designer has this freedom, information is given in Sec. II which enables the designer to help specify the proper error-correcting code to be used for representing the data, both within and without the memory system. There is some evidence that asymmetric codes, that is, codes which can be used for correcting or masking open-circuit failures but not short-circuit failures, can be used for economical read access switches. It is suggested that further work on reliable read access switches be directed along this line.

Write access switches which are tolerant of certain failures both of the inhibit drivers and of the incoming code word were briefly treated in Sec. III. The example which was considered seemed to indicate that such switches can be designed, although the problem appears to be somewhat more difficult than that of read access switch design. It is clear that for a more general treatment than was attempted in Sec. III, the linear separability problems of the codes used must be considered in some detail. This is by no means a trivial problem; however, a body of literature does exist in this field and some of it may be found to be relevant.

If single apertured cores are used for the memory cells, the read and write access switches which were treated in Secs. II and III can be used only if the currents in their secondary circuits are properly limited. Two methods for accomplishing this were considered in Sec. IV. One was the use of a large load-sharing factor, and the other was the use of a passive non-linear magnetic load. It appears that currents may be limited to a few percent using practically-realizable parameters in such a magnetic device, and without an undue wastage of power. The magnetic equations for this current limiter were written, using a simple model, and a design method was developed. Clearly, further work is needed on this device, both on developing a set of design equations based on a more accurate magnetic model and on experimentally verifying the results.

Two portions of the memory system have not been treated in this part of the report in any detail: these are the protection of the sense amplifiers, and the protection of the memory cells. It appears that in further work these two areas should be considered. In particular, an application of some of the published techniques of multiple coincidence[2, 3, 4, 5] appears to be appropriate to both of these problems.

Finally, it should be pointed out that while a number of techniques have been developed for protecting various portions of a memory system, a number of problems are likely to arise when these techniques are incorporated simultaneously in a memory system. Therefore, it appears that in further work on this topic consideration should be given not only to the development of several additional reliable memory techniques, but also to the practical problems of assemblying these techniques into an operating reliable memory system.

VI REFERENCES FOR PART TWO

1 Hunter, L. P., and Bauer, E. W., "High Speed Coincident-Flux Magnetic Storage Principles," *J. Appl, Phys. 27(11)*, pp. 257-261, (November 1956).

2 Minnick, R. C., and Ashenhurst, R. L., " Multiple Coincidence Magnetic Storage Systems," *J. Appl. Phys. 26*, pp. 575-579, (May 1959).

3 Ashenhurst, R. L., "The Structure of Multiple Coincidence Selection Systems," Doctoral Thesis, Division of Engineering and Applied Physics, Harvard University, Cambridge, Massachusetts, (May 1956).

4 Schlaeppi, H. P., and Carter, I. P. V., "Submicrosecond Core Memories Using Multiple Coincidence," *IRE Trans. EC-9*, p. 132, (June 1960).

5 Blachman, N. M., "On the Wiring of Two-Dimensional Multiple-Coincidence Magnetic Memories," *IRE Trans. EC-5*, pp. 19-21, (March 1956).

6 Kautz, W. H., "Codes and Coding Circuitry for Automatic Error Correction Within Digital Systems," Tech. Report 2, SRI Project 3196, Stanford Research Institute, Menlo Park, California (January 1962).

7 Haynes, J. L., and Minnick, R. C., "Magnetic Core Access Switches," RADC TR-61-117B, SRI Project 3184, Stanford Research Institute, Menlo Park, California (May 1961).

8 Karnaugh, M., "Pulse Switching Circuits Using Magnetic Cores," *Proc. IRE 43*, pp. 570-583 (May 1955).

9 Olsen, K. H., "A Magnetic Matrix Switch and its Incorporation into a Coincident-Current Memory," MIT DCL Report R-211, Digital Computer Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, (June 1952).

10 Takahashi, H., and Goto, E., "Application of Error-Correcting Codes to Multi-Way Switching," *Proceeding of the International Conference On Information Processing*, pp. 396-400 (UNESCO, Paris, 1959).

11 Petersen, W. W., *Error Correcting Codes*, (John Wiley and Sons, Inc., New York, N. Y., 1961).

12 Stuart-Williams, R., "Magnetic Switching System," U. S. Patent 2,691,152 (October 5, 1954).

13 Mann, H. B., *Analysis and Design of Experiments*, (Dover Publications, Inc., New York, N. Y., 1949).

14 McNaughton, R., "Unate Truth Functions," *IRE Trans. EC-10(1)*, pp. 1-6 (March 1961).

PART THREE

LOGICAL REDUNDANCY TECHNIQUES*

I INTRODUCTION

The general long-range objective of this part of the project effort is concerned with redundancy techniques for digital systems on the detailed, logical level, and calls for both the development of new redundancy techniques and the beginnings of design philosophy for redundant digital networks. The first year of the project was occupied mainly with techniques for combinational circuitry, while sequential networks have been the focus of attention during the second year.

The major detailed technical results of this portion of the project have been communicated to the sponsor in two Technical Reports issued earlier:

No. 1. "Automatic Fault Detection in Combinational Switching Networks," by William H. Kautz, April 1961.

No. 2. "Codes and Coding Circuitry for Automatic Error Correction within Digital Systems," by William H. Kautz, January 1962.

Abstracts of these reports are included in Appendix A of this part of the present report. Both have since appeared in the technical literature.[1,2**] In addition, other results have been presented in past quarterly letter reports. This final report consists of a summary and evaluation of the project activity in the area of redundancy techniques on the detailed, logical level.

The bibliography on redundancy techniques, which has been maintained up-to-date throughout the project, was checked in January 1962 against a similar bibliography by Paul A. Jensen. Jensen's bibliography, including numerous additions from ours, has been published in the Proceedings of the recent Conference on Redundancy Techniques in Computing Systems.[3] Several recently discovered references that are not contained in Jensen's final list are collected into Appendix B of this part of the report.

*by William H. Kautz

** References are grouped at the end of Part Three.

II FAULT DETECTION WITH CIRCUIT REDUNDANCY

The principal results in fault detection consist of the new redundancy procedures themselves, as described in Technical Report 1 for combinational circuitry, and in Quarterly Letter Reports 3 (pages 5 to 7) and 6 (pages 2 to 6) for sequential circuits. These procedures apply to both single-output and multi-output digital networks, composed of either gate-type or branch-type logical elements. In the combinational case, several examples of circuitry in application areas of direct practical importance are worked out. In all cases, it is shown that the redundancy ratio for single fault detection never need exceed about 2. In many cases it can be made much less by taking advantage of inherent redundancy already present in an irredundant realization or in the set of output functions themselves, or by limiting the protection to only the most likely types of faults, possibly detected with some delay in the indication of the fault.

Very little has been previously published on the use of circuit redundancy for fault detection, aside from an implicit reliance on simple duplication with comparison, and the use of loop checks for certain easily invertible digital operations.

The principal use of fault detection in unmaintained digital systems, such as space craft, is for the monitoring and controlled switch-in of redundant spares. With the aid of fault detection, faults may be sufficiently localized to allow defective networks and subsystems to be automatically switched out of the system, and electrically replaced with fresh units. Thus, fault detection, plus spares with switch-in capability, constitutes one form of automatic self-repair.

A variety of techniques is therefore available to the logical designer for introducing circuit redundancy for fault detection in combinational circuits. These techniques, or small modifications of them, will certainly cover any reasonable design situation which is at all likely to arise with conventional logical elements. For sequential circuitry, however, we must conclude that the state-of-the-art does not allow such an optimistic claim. While the techniques offered will undoubtedly be adequate for many problems, they can serve as only a guiding principle in others, and perhaps not even that much in still others. Except for some specific logical elements and specific operations, this situation will probably not be very much improved until an improved mathematical theory is available for the analysis and synthesis of general, irredundant sequential networks. This conclusion is consistent with an earlier observation which appears to hold in general--namely, that it is very difficult to develop techniques for the synthesis of redundant switching networks of any particular type until synthesis techniques have first been developed for the corresponding irredundant networks.

It may also be concluded that fault detection techniques should be applied on the level of the network or higher, and in no cases on the level of the component or elementary logical element. The choice of the exact, preferred level within a digital system will depend on how the required redundancy ratio varies with the level, and also on the possible use of loop checking or signal redundancy for fault detection, and on the type and extent of switching required for the connection of spares.

One of the redundancy techniques proposed for use with sequential networks involves the use of *unit-distance* counting sequences. These are codes such that the passage from each state (or count) to the next requires a change in only a single binary digit. When error-checking features are added to these codes, they become useful not only for fault detection in sequential circuits, but in analog-to-digital converters as well. Two families of such codes are presented in Quarterly Letter Report 6 (pages 2 - 6), and the circuit equations for some related operations are derived: counting, adding, conversion to binary codes, and error detection. These codes should find direct application in the design of economical self-checking analog-to-digital conversion equipment.

A theoretical question raised in Technical Report 1 (page 12), and answered at the time only for simple functions, concerns the possibility of reducing the cost of a certain checking network for detecting Type I and II AND-gate faults in AND-OR networks. We have now been able to show that such a simplification is never possible, even for very complex functions. In switching theoretic terms, the proof consists of showing that the complexity of the reduced-normal-form realization of a unate function can never be reduced if the uncomplemented minterm $x_1x_2 \cdot \ldots x_n$ is removed from the function.

Several specific problems may be singled out for further study. For combinational networks, it would be useful for certain applications to have synthesis procedures that take advantage of unspecified input-variable combinations (so-called "don't cares") to simplify the network realization. While such procedures are available for irredundant networks, they need to be worked out for redundant structures as well. It would also be helpful to have a better description of just which switching operations require redundancy ratios well below the upper limit of 2, so that these operations may be preferred during a system design. Similarly, it may be possible to evolve some way of determining the redundancy ratio required for a given function, network form, and type of fault without actually executing in detail the entire redundant design.

More specific procedures might be worked out for networks composed out of less conventional logical elements, such as magnetic or threshold elements, at least for some common networks such as adders, comparators, etc., if not for networks in general.

In the sequential area, both irredundant and redundant procedures are needed for arbitrary networks. In view of the difficulty of this general problem, it may be more advisable to concentrate for the time being on specific network structures and functions, and on some specific types of logical elements of current importance.

For analog-to-digital conversion, it may be possible to evolve some other error-detecting codes which are more efficient than those proposed in Letter Report 6, but which lead to no more complex logical circuitry for adding, counting, code conversion, etc. Similar families of codes for error *correction* could also be developed.

III FAULT MASKING WITH CIRCUIT REDUNDANCY

A large number of fault-masking techniques have been proposed in the technical literature, some of them complete with evaluations showing statistically the improvements in reliability which are possible.[4-13] Although usually proposed for use at the level of the component or logical element, most of the techniques are also applicable at higher levels within a digital system. Also, they can usually be applied to sequential circuits, even though specifically proposed only for combinational circuits.

Unfortunately, the redundancy ratio which results from the use of these techniques is very high, reaching a minimum of 3 or 4 only for the simplest approaches--those due to von Neumann and to Tryon for gate-type elements, and to Moore and Shannon for branch-type elements. No advantage is taken even in these cases of any pre-existing logical redundancy in the irredundant realization, or of the possibility of masking only the most likely types of faults.

Our studies in fault-masking on the network level for combinational circuits indicated that there exist single-output networks which may be rendered insensitive to single faults at a cost well below this minimum ratio of 3 or 4. Several such examples were produced, although the particular approach taken did not evolve any simple definition of the entire class of network functions for which such simplification is possible. Similar and sometimes even greater gains are possible in multi-output networks, because of the possibility of inherent redundancy in the output function specifications. For example, the disjunctivity of the outputs of a gate-type complete decoding network can be used to advantage for fault masking at a redundancy ratio somewhat less than 3. Similar results hold for complementary-output gate-type and branch-type networks.

This fault-masking approach for combinational networks is discussed in Letter Report 4 (pages 1 - 12), and leads to the conclusion that for this family of networks, fault masking is usually but not always expensive. The best hopes for more economical fault masking appear to lie in the use of signal redundancy, and in application of logical redundancy at a high level--that is, to large networks and large portions of the digital system. The successful execution of the latter alternative requires a greater understanding of how logical redundancy can be determined and isolated in arbitrary logical functions of high complexity, and how an irredundant network may be redesigned to incorporate the minimal additional redundancy which is required for masking single faults. Particular emphasis should be placed on multi-output networks, since these potentially contain more inherent redundancy in their functional specifications.

Fault-masking can be applied to sequential networks through separate checking of the logical and storage portions of the network, as described for fault detection in Letter Report 3 (pages 5 - 7), and in Technical Report 1 (pages 27 - 33) for iterative networks, or by Lofgren's method, but based on the structure of an error-*correcting* code instead of an error-detecting code. For large networks, the cost will be less than that which would result from triplication (von Neumann) or quadruplication

(Tryon) but the design will likely be inefficient, because of the necessity of realizing the multi-output logic network as separate single-output networks. Such a decomposition is not basically necessary, but must be resorted to until synthesis procedures for redundant multi-output combinational networks are available.

In conclusion, then, fault-masking techniques for circuit redundancy are available for all types of networks, but are costly. This cost could probably be reduced for most large networks and for a few small networks through the development of improved synthesis techniques.

IV SIGNAL REDUNDANCY

The numerous techniques described in Technical Report 2 should provide the logical designer with a broad spectrum of competence for introducing redundancy into a digital network, subsystem, or system via the data or *signals* which pass through it. Errors which result from faults may be either detected or corrected, depending on whether the code on which the signal redundancy is based is interpreted as an error-detecting or an error-correcting code. Any of these signal redundancy techniques may be applied to the checking of data storage and transfer operations. A few techniques are offered for checking arithmetic operations, although these are not fully developed, and may not be considered seriously competitive with simple duplication or triplication of serial arithmetic units. Other techniques yield a small saving for limited fault types, such as those faults which give rise to asymmetric errors.

The techniques which are offered apply to both serial and parallel transfer and storage of data, although different codes are usually preferred in the two cases. For the most part, only single errors are corrected, although multiple error detection is sometimes convenient. The study was most successful for the series-parallel channel, for which the required amount of redundant circuitry is quite modest for an attractive combination of single-error correction plus multiple-error detection. The effectiveness and efficiency of this type of code results from the combination of the advantages of both serial (sequential) and parallel (combinational) codes: Each checks the weaknesses of the other, and also checks a part of the susceptibilities of the channel to cause errors. Thus, the parallel code corrects burst-type, asymmetric errors which result from serial handling of the data, once the particular parallel channel that is defective is identified. At the same time, the serial code serves to isolate this parallel channel.

Additional future work is justified on the optimal use of series-parallel codes in conjunction with specific memory types and register types.

In general, the error-correcting portions of the redundant system are the most susceptible remaining parts. Some limited techniques, one of them based on the iteration of error-correcting codes, were proposed for this purpose, but an additional effort should be made to determine how to best correct or mask faults in these correctors. Faults in the encoders normally appear at the decoder just as if they had occurred in the channel (memory, register, or transfer circuitry), so they need not be treated specially.

Finally, better codes are needed for checking arithmetic operations, with emphasis on detecting or masking only those types of faults which are most likely to arise.

V REFERENCES FOR PART THREE

1. Kautz, W. H., "Automatic Fault Detection in Combinational Switching Networks," in *Switching Circuit Theory and Logical Design*, pp. 195-214, AIEE Publication S-134 (American Institute of Electrical Engineers, New York, New York, 1961).

2. Kautz, W. H., "Codes and Coding Circuitry for Automatic Error Correction within Digital Systems," in *Redundancy Techniques for Computing Systems*, pp. 152-195, ed. by R. H. Wilcox and W. C. Mann (Spartan Books, Washington, D.C., 1962).

3. Jensen, P. A., "Bibliography on Redundancy Techniques," in *Redundancy Techniques for Computing Systems, loc. cit.*, pp. 389-403.

4. Von Neumann, J., "Probablistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies, Annals of Mathematics Studies*, No. 34, pp. 43-49 (Princeton University Press, Princeton, New Jersey, 1956).

5. Tryon, J. G., "Quadded Logic," in *Redundancy Techniques for Computing Systems, loc. cit.*, pp. 205-228

6. Moore, E. F. and C. E. Shannon, "Reliable Circuits Using Less Reliable Relays," *J. Franklin Inst., 262*, pp. 191-208 (September 1956); and pp. 281-297 (October 1956).

7. Brown, W. G., J. Tierney, and R. Wasserman, "Improvement of Electronic Computer Reliability Through the Use of Redundancy," *IRE Trans., EC-10*, pp. 407-416 (September 1961).

8. Winograd, S. and J. D. Cowan, "Minimally Redundant Reliable Computing Systems Design," in *Redundancy Techniques for Computing Systems, loc. cit.*, p. 377.

9. Amarel, S. and J. A. Brzozowski, "Theoretical Considerations on Reliability Properties of Recursive Triangular Switching Networks," in *Redundancy Techniques for Computing Systems, loc. cit.*, pp. 70-128.

10. Kilmer, W. L., "An Idealized Over-all Error Correcting Digital Computer Having Only an Error Detecting Combinational Part," *IRE Trans., EC-8*, No. 3, pp. 321-325 (September 1959).

11. Lofgren, L., "Irredundant and Redundant Boolean Branch Networks," *IRE TRANS., CT-6*, pp. 158-175 (May 1959).

12. Lofgren, L., "Automata of High Complexity and Methods of Increasing Their Reliability by Redundancy," *Inform. and Control, 1*, No. 2, pp. 127-147 (1958).

13. Armstrong, D. B., "A General Method of Applying Error Correction to Synchronous Digital Systems," *Bell Syst. Tech. J., 40*, 2, pp. 577-593 (March 1961).

APPENDIX A

ABSTRACTS OF TECHNICAL REPORTS 1 AND 2

APPENDIX A

ABSTRACTS OF TECHNICAL REPORTS 1 AND 2

I TECHNICAL REPORT 1 (AUTOMATIC FAULT DETECTION IN COMBINATORIAL SWITCHING NETWORKS)*

This report is concerned with the logical design of economical combinational switching networks which contain sufficient redundancy so that the presence of any single fault can be detected.

It is well known that (within broad limits) any isolated fault in a single-output network may be detected with about 2:1 redundancy, through duplication of the irredundant network, and the addition of a comparator gate. In the procedures to be described, a reduction of the redundancy ratio below 2:1 is shown to be usually possible, by (1) taking advantage of any inherent logical redundancy in the switching function or functions which describe the terminal behavior of the network; (2) making use of any structural redundancy present in the irredundant version of the network; (3) providing for the detection of only those faults considered to be at all likely to occur, rather than "all" faults categorically; (4) where possible, allowing some faults to be detected with a delay, rather than at the first occurrence of an erroneous output; and (5) application of certain principles of error-detecting codes. A similar reduction of the redundancy ratio is possible for multi-output networks.

Branch-type networks--*e. g.*, those made up of relay contacts or cryotrons instead of gate-type elements--are also considered. If the designer is free to use some non-branch-type elements in the detection circuitry, the same low redundancy ratios achievable in gate-type circuitry can be obtained here.

Finally, several examples in application areas of importance in conventional digital systems are discussed, including some comparators, calculating networks, code converters, and decoding trees.

---

* by William H. Kautz

II TECHNICAL REPORT 2 (CODES AND CODING CIRCUITRY FOR AUTOMATIC ERROR CORRECTION WITHIN DIGITAL SYSTEMS)*

Error-correcting codes for use in communication channels have been known for many years. Recently discovered code families have enabled economical encoding and decoding equipment to be designed for use with even very noisy channels. These new codes provide for the automatic correction of several independent errors in a block of transmitted digits, or of bursts of errors in successive digits.

Within a digital computer the "channel" becomes the collection of data paths and storage locations embodied in index, data, and control registers and in the memories and associated input-output transfer links. An attempt to apply existing error-checking codes to this type of channel reveals that these known codes are not necessarily optimal, because (1) appropriate measures of efficiency may differ from those customary in the traditional communications channels, (2) the most probable types of errors (noise) may not be the same, (3) one must also take into account the possibility of errors due to faults in the decoding logic (corrector) itself, and (4) if possible, the codes should be compatible in some sense with arithmetic and other related operations.

In this report we discuss the efficiency, cost, and optimality of codes for use within digital systems, and evaluate to what extent known code families can be applied to automatic error correction. To better satisfy these criteria in the most important remaining cases, some new codes are then described. Particular attention is devoted to error correctors which are "fault-masked"--*i. e.*, whose performance is insensitive to single isolated circuit faults. Also discussed are the simplifications possible if mere detection of an error is adequate. Several examples of encoding, corrector, and detector circuits are offered, exemplifying the use of both gate-type and branch-type logical elements.

* by William H. Kautz

APPENDIX B

SECOND ADDENDUM TO BIBLIOGRAPHY ON THE IMPROVEMENT OF RELIABILITY THROUGH REDUNDANCY

APPENDIX B

SECOND ADDENDUM TO BIBLIOGRAPHY ON THE IMPROVEMENT OF RELIABILITY THROUGH REDUNDANCY

Christian, J., and Hollander, K. W., "Reliability of Switching Mechanisms," Report RADC-TR-60-239, Contract AF 30(602)-2143, Project 5019, Task 45155, ARINC Research Corporation, Washington, D. C., (January 1961).

Fleck, J. J., et al, "Research and Feasibility Study to Achieve Reliability in Automatic Flight Control Systems," WADD Tech. Report 61-264, Contract AF 33(616)-7260, Flight Control Laboratory, Wright Air Development District, Air Research Development Command, United States Air Force, Wright-Patterson Air Force Base, Dayton, Ohio (April 1961).

Griesmer, J. H., Miller, R. E., and Roth, J. P., "The Design of Digital Circuits to Eliminate Catastrophic Failures," Research Report RC-604, IBM Research Center, Yorktown Heights, New York (January 1962).

Klass, P.J., "IBM System Chosen for OAO Satellite," *Aviation Week 74*, No. 21, p. 83 (22 May 1961).

Liddell, D. W., "Integration and Automatic Fault Location Techniques in Large Digital Data System," *Proc. SJCC*, pp. 213-224 (National Press, Palo Alto, California, 1962).

Lofgren, L., "Qualitative Limits for Automatic Error Correction Self-Repair," Tech. Report 6, University of Illinois Electrical Engineering Research Laboratories, Urbana, Illinois (June 1960).

Lofgren, L., "Self-Repair as a Computability Concept in Automata Theory," *Proc. of Symposium on Mathematical Theory of Automata*, (Polytechnic Institute of Brooklyn, In press).

Lofgren, L., "Self-Repair as the Limit for Automatic Error Correction," *Proc. of Symposium of Self-Organizing Systems*, (Pergamon Press, New York, N. Y., 1960).

Lofgren, L., "Kinematic and Tessellation Models of Self-Repair," Tech. Report 8. University of Illinois Electrical Engineering Research Laboratories, Urbana, Illinois (December 1961).

Maitra, K. K., "Stability of Logical Networks and Its Application to Improvement of Reliability," *IRE Trans. CT-9*, No. 3, pp. 335-341 (September 1961).

Pooge, J. F., "Derivation of Optimum Tests to Detect Faults in Combinatorial Circuits," *Proc. Symposium on Mathematical Theory of Automata*, (Polytechnic Institute of Brooklyn, In press).

Schwartz, L. S., and Hilton, A. M., "Probability, Statistics, and the Theory of Games," *ElectroTechnology 68,* pp. 101-124 (March 1961).

Toeste, R., "Reliability and Redundant Computers," Report 21-G-0029, Lincoln Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts (22 March 1961).

Verbeek, L. A. M., "Note on the Synthesis of Infallible Networks," Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts (15 April 1961).

Weiss, G. H., and Kleinerman, M. M., "The Reliability of Sequentially-Operated Networks," Report 7273, U. S. Naval Ordinance Laboratory, (White Oak, Maryland, Oct. 1960).

Wilcox, R. H., and Mann, W. C., *Redundancy Techniques for Computing Systems,* (Spartan Books, Washington D. C., 1962).

Wouk, A., "On Optimization of Faulty Channels," *Soc. Ind. Appl. Math. J. 9,* No. 2, pp. 311-316 (June 1961).